OS/2 Warp 多媒体子系统
编 程 指 南

[美] IBM 公司 著

刘晓玲 黄 俊 陈 雄
黄永刚 向红军 译

彭 丰 审校

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 提 要

该书是 IBM 公司授权清华大学出版社独家翻译出版的一系列 OS/ 2 Warp 使用及开发手册中的一本。它详细介绍了构成多媒体子系统应用程序的媒体控制驱动程序、流处理器、I/ O 过程的结构、特点、内容及安装要求,非常适合于有兴趣开发 OS/ 2 多媒体子系统的编程人员。

该书实用性强,内容准确,叙述通俗易懂。同时给出了大量的编程实例,生动形象地指导编程人员进行开发工作,不失为编程人员的良师益友。

图书在版编目(CIP)数据

OS/ 2 Warp 多媒体子系统编程指南/ 美国 IBM 公司编;刘晓玲等译 .—北京:清华大学出版社,1996

ISBN 7-302-02204-6

Ⅰ .0… Ⅱ .①美… ②刘… Ⅲ .多媒体技术-子程序,OS/ 2 Warp-程序设计-指南 Ⅳ .TP311-62

中国版本图书馆 CIP 数据核字(96)第 09167 号

出版者: 清华大学出版社 (北京清华大学校内,邮编 100084)
印刷者: 北京市海淀区清华园印刷厂
发行者: 新华书店总店北京科技发行所
开 本: 787×1092 1/ 16 印张: 21 字数: 497 千字
版 次: 1996 年 9 月 第 1 版 1996 年 9 月 第 1 次印刷
书 号: ISBN 7-302-02204-6/ TP·1061
印 数: 0001—4000
定 价: 36 .00 元

OS/ 2 Warp Multimedia Subsystem Programming Guide

First Edition (October 1994)

出版国外(海外)图书合同登记号: 图字: 01-96-0135 号

第一版(1994 **年** 10 **月**)

本书可能会出现技术或排版印刷的错误。因此,IBM 公司会定期修订此书,并将修订后的内容纳入新版本中。IBM 也将随时改进并(或)变动本书中所提及的产品及(或)程序。

本书可能会提及或引用本国尚未推出的 IBM 产品(软,硬件),程序或服务项目,这并不意味着 IBM 有意在本国推出这些产品、程序或服务项目。

若需 IBM 产品的技术信息,请洽询 IBM 所授权的经销商或业务代表。

目　　录

关 于 这 本 书

本书为多媒体子系统开发指南。每一子系统的各部分将在不同章节中详细介绍。各部分的样例程序模板用于辅助指导。

1．相关出版物

以下列出了 OS/ 2 Warp 的技术库丛书:

· OS/ 2 Warp 技术库:

控制程序编程指南
控制程序编程参考
图形编程接口编程指南
图形编程接口编程参考

信息显示设备编程指南
多媒体应用程序编程指南
多媒体编程参考
多媒体子系统编程指南

显示管理器编程指南(高级)
显示管理器编程指南(初级)
显示管理器编程参考
REXX 参考

REXX 用户指南
工具参考
工作区外壳编程指南
工作区外壳编程参考

· 关于 OS/ 2 的 IBM 设备驱动程序的出版物:

显示设备驱动程序参考
输入/ 输出设备驱动程序参考
MMPM/ 2 设备驱动程序参考
OS/ 2 的笔式设备驱动程序参考
物理设备驱动程序参考
显示驱动程序参考

打印设备驱动程序参考

存储设备驱动程序参考

虚拟设备驱动程序参考

2. 附属的多媒体信息资料

· Mutimedia REXX-(联机手册)

书中详细介绍了 REXX 的功能,这些功能将多媒体控制接口字符指令从 OS/2 命令文件传送出来,以控制多媒体设备。本书由 OS/2 多媒体自身提供。

· 多媒体用户接口设计指南

书中详细介绍了在设计通用用户入口(CUA)多媒体界面时要考虑到的设计思想,就是要保证单件多媒体产品与其它产品的兼容性。

· OS/2 多媒体设备驱动程序开发

书中摘录了编辑过的源代码及其它关于怎样为 OS/2 多媒体环境安装虚拟设备驱动程序(VDD)和音频物理设备驱动程序(PDD)的详细资料。工具箱包含了所有 OS/2 物理设备驱动程序和虚拟设备驱动程序:有打印机,显示器,SCSI、CD-ROM 驱动器及适用于 OS/2 多媒体环境的录入器等。其中还有所有设备驱动程序的界面和系统服务的综合论述。

第 1 章　多媒体子系统概论

OS/ 2 多媒体(指早先发行的多媒体显示管理器/ 2 或 MMPM/ 2)拥有可扩展结构,以便随着多媒体技术的发展,加入新的功能、设备及多媒体数据格式等。本章综述 OS/ 2 多媒体系统中的各个系统部件。随后的各章节将详细指导读者通过图 1-1 所示的样例程序安装和开发个人的 OS/ 2 多媒体子系统。根据你对多媒体的需求,可以方便地对样例程序模板进行修改,也可以用来为 OS/ 2 环境安装和开发多媒体控制驱动程序、流处理器及 I/ O 过程。

图 1-1　样例程序子目录结构

1.1　OS/ 2 多媒体系统结构

图 1-2 展示了 OS/ 2 多媒体系统各子系统部件。在 OS/ 2 环境下,这些子系统包括多媒体控制驱动程序、流处理器及 I/ O 过程,并且均由程序管理器控制、监视其一系列工作。

在第三层环上,OS/ 2 多媒体使用了多媒体设备管理器 MDM,用于管理逻辑多媒体设备,例如声卡,CD-ROM 驱动器等其它硬件设备。在 MDM 的所有工作中,有一项就是:当有两个或两个以上应用程序申请控制同一台多媒体设备时,由它决定选择哪个应用

图 1-2　多媒体显示管理器/ 2 系统结构

程序取得控制权。

同步/ 流管理器(SSM),同样可以在多媒体控制驱动程序初始化后,用来管理流和同步调用。这样,就可以取消每个多媒体驱动程序对公用的多媒体设备的需要,而由本身找到解决办法。多组流处理器完成从源设备到目的设备的数据传输,而由 SSM 统一协调,并对数据缓冲区和同步数据进行集中管理。

最终由 MMIO 管理器完成诸如多媒体控制驱动程序、应用程序等子系统部件对各种数据的接收和操作,这些数据有图象、图表、数字音频信号、数字视频信号等等。通过不同的存储系统,不同数据以不同的文件格式存储,由 MMIO 管理器通过可安装的 I/ O 过程来完成与读写操作有关的、对不同存储类型和文件表格进行的输入输出操作。

1.2 可扩展设备支持

OS/2 多媒体系统结构可以扩展。模块化结构设计允许对新发展的硬盘设备、逻辑设备、文件格式等的支持。

多媒体控制接口设备的示例参见表 1-1。此表提供的是系统能支持的、并且已被多媒体控制接口定义过的逻辑设备类型。打勾表示的是 OS/2 直接支持的设备。

表 1-1 多媒体控制界面逻辑设备类型

多媒体设备类型	OS/2 多媒体	字符串	通用文件名
混响放大器		ampmix	MCI_DEVTYPE_AUDIO_AMPMIX
磁带唱机		audiotape	MCI_DEVTYPE_AUDIO_TAPE
CD 音频演播器		cdaudio	MCI_DEVTYPE_CD_AUDIO
CD-XA 演播器		cdxa	MCI_DEVTYPE_CDXA
数字式录音机		dat	MCI_DEVTYPE_DAT
数字式录象机		digitalvideo	MCI_DEVTYPE_DIGITAL_VIDEO
耳机		headphone	MCI_DEVTYPE_HEADPHONE
麦克风		microphone	MCI_DEVTYPE_MICROPHONE
监视器		monitor	MCI_DEVTYPE_MONITOR
其它		other	MCI_DEVTYPE_OTHER
视频覆盖		videooverlay	MCI_DEVTYPE_OVERLAY
音序器管理器		sequencer	MCI_DEVTYPE_SEQUENCER
扬声器		speaker	MCI_DEVTYPE_SPEAKER
视盘演播器		videodisc	MCI_DEVTYPE_VIDEODISC
录象机		videotape	MCI_DEVTYPE_VIDEOTAPE
波形音频演播器		waveaudio	MCI_DEVTYPE_WAVEFORM_AUDIO

注意:M-Control Program Version 2.01 支持 M-Motion Video Adapter/A,并提供OS/2 多媒体覆盖技术。

1.3 多媒体控制驱动程序

多媒体控制接口提供应用程序控制多媒体设备的主要结构。最顶层包含 MDM,由它对多媒体设备进行最初管理;最底层包含各种多媒体驱动程序(MCD)——动态链接库实现对设备的控制。如图 1-3 所示。

应用程序与多媒体控制接口,继之与多媒体设备之间的相互作用有两种方式:控制程序接口(mciSendCommand)和字符串接口(mciSendString)。但是,在 MCD 解释字符命令之前,必须通过查询命令表来完成从字符命令到相应控制程序命令之间的转换。

图 1-3　多媒体控制接口结构

MCD 并不直接控制硬件设备,而是通过命令在子系统或物理设备驱动程序接口的传输来实现。这种设计将 MCD 从必须确认硬盘以完成其功能中解脱出来。不过,MCD 不得不熟悉完成那种功能的方法。举例说,一台 CD-Audio 播放器通过使用 CD-ROM 驱动器及其内部的数-模转换器(DAC),靠简单地分配设备 IOCtl 指令到驱动设备程序中,就能完成。只不过与前面不同的是,一台 CD-Audio 播放器采用独立的数字信号处理器(DSP)记录数字音频数据,采用功能调用实现对 DSP 协处理器的管理及数据在驱动器和协处理器的传输管理。

MCD 也可以使用其它多媒体子系统提供的服务,例如流编程接口(SPI)等。这种系统提供数据流服务,即就是允许流处理器控制设备之间的实时数据流,维持物理设备之间的数据连续流动。

1.4　I/O 控制程序过程

多媒体输入输出(MMIO)服务具有 I/O 和 CODEC 两种控制程序。I/O 控制程序是基于信号的处理程序,直接控制在不同的存储系统和文件表之间与读写操作有关的输入输出操作。通过 MMIO 信号将应用程序和 MMIO 子系统联系起来。当 MMIO 接收到来自应用程序的功能调用指令时,MMIO 管理器就送出与指令相适应的预先定义好的信息给 I/O 过程,负责对特定的文件表或存储系统进行操作。同样,I/O 过程按照 MMIO 管理器或应用程序发出的指令进行操作。

上述信息的设计实现所有 I/O 过程之间的相互有效联接。当然,I/O 过程也必须能处理信息或能传送信息给其子程序。例如,I/O 过程接收到要求数据压缩的信息,则必须

能够处理这条信息或将其传到 CODEC(编码/ 译码)程序。图 1-4 所示的就是 MMIO 子系统中 I/ O 和 CODEC 程序之间的相互关系。

图 1-4　多媒体 I/ O(MMIO)子系统结构

MMIO 管理器调用的子程序有:

1 . 文件表

文件表子程序是用来对数据进行操作的 I/ O 程序,其每一子程序处理不同类型元素,如音频、图象、MIDI 等,并且能够单独处理数据而不依赖其它子程序。但是,文件表子程序可能需要调用存储系统 I/ O 过程,以便从含多个元素的文件中取得数据。

2. 存储系统

存储系统子程序是一种相当于为文件表程序解开数据并取出来的 I/ O 过程。在上述过程中,存储系统将忽略数据表的格式。

3. 编码/ 译码子程序(CODEC)

CODEC 子程序对文件或缓冲区数据进行操作。根据数据的内容,I/ O 过程可以装入 CODEC 子程序用于压缩或解压缩数据。

1.5　流 处 理 器

多媒体系统可以在系统内核级(第 0 环)或应用程序级(第 3 环)提供流处理器。流处理器安排在这两级是因为许多流理论上在与设备的物理驱动程序(PDD)直接联系时,要受到控制;其它的流则不与源数据或目标数据联系,而物理上直接与特定设备联系。例如,文件系统流处理器是 DLL,由于所有文件系统的 I/ O 功能可在第 3 环 OS/ 2 功能中得到,并为所有文件系统设备服务,这就不用为每一个文件系统进入而建立特定的流处理设备驱动程序。

流处理器负责对应用数据的连续、实时方式的流动实现控制。每一处理程序可以建立多个数据流层,其中每个流包含特定类型的数据,如 MIDI(音响设备数字接口)或 ADPCM(自适应增量脉冲编码调制)等。应用程序通过使用媒体控制驱动程序,调用 SPI 功

图 1-5　同步/ 流子系统结构

能来产生流或调用其它 SPI 功能来激活数据流,而不必一直调用 SPI 功能来维持数据流。相反,流处理器能保持 I/ O 连续性,并简化应用程序的操作。

第 2 章　媒体控制驱动程序

媒体控制驱动程序(Media Control Driver 缩写为 MCD)。本章描述了各种流式和非流式的 MCD 的编程接口,并借此来告诉你如何编写 MCD。下面将把讨论的重点放在样例程序二重唱演奏器(Duet Player)中所用的 MCD(详见《OS/2 多媒体应用程序编程指南》一书)。波形音频 MCD 通过 SPI 数据流方式来创建和管理源数据流处理器和目标数据流处理器。而 CD 音频 MCD 则不然,它们无需相关的数据流,因为大多数的 CD 音频设备的数据处理是内部完成的(即相对于本设备是内部的,无需求助于主 CPU)。

在 \ TOOLKIT \ SAMPLES \ MM 子目录中,可以找到下列媒体控制驱动程序样例的源代码。

1 . 媒体控制驱动程序模板(MCDTEMP)

提供了一个编写 MCD 的基本模板。而对于具体的流式或 MMIO 的样例,参阅子目录 ADMCT 和 CDMCIDRV。

2 . 波形音频媒体控制驱动程序(ADMCT)

提供了一个关于怎样控制流式设备的例子。流式设备使用 OS/2 多媒体的同步/流管理器(Sync/Stream Manager 缩写为 SSM)来控制数据流从源位置流向目标位置。例如,二重唱演奏器 I 中所用的声卡就是一种流式设备,它用来播放存在用户的硬盘上的波形音频文件。PM 应用程序向媒体设备管理器(Media Device Manager,简写为 MDM)发出播放这些文件的请求。然后,MDM 调用波形音频 MCD,而后者使用流处理器将盘上的波形文件(源)中的数据放入声卡(目标)上的缓冲区。

3 . CD 媒体控制驱动程序(CDMCIDRV)

提供了一个关于怎样控制非流式设备的例子。非流式设备在设备内部传递数据流。因为数据的源和目标都在设备内部,所以此类设备可以在内部传递数据流,而无需使用具有缓冲区的 I/O。因此,一个非流式设备也就不要求使用 OS/2 多媒体的 SSM 子系统。例如,二重唱演奏器Ⅱ中所用的 CD-ROM 驱动器就是一个非流式设备,当 CD 音频 MCD 从程序中收到 PLAY,PAUSE,或 STOP 命令后,它向 CD-ROM 驱动器发出相应的 IOCtls 来实现这些动作。在这里硬件包揽了所有的工作:从光盘读取数字化的信息,把它翻译成音频信号并送往某个输出端口,比如耳机插座等。

2 .1　媒体控制驱动程序的体系结构

多媒体应用程序使用 mciSendCommand 或者 mciSendString 来发出设备控制命令,而媒体控制驱动程序就是这些命令的接收者。媒体控制驱动程序是一些 OS/2 的动态链接库(DDL),它可在一定程度上使应用程序具有设备无关性;而媒体设备管理器(MDM)则提供了通向 MCD 的接口、另外,MDM 还具有一定程度的局部管理的功能,这使一个应用

程序可以同时使用不同的 MCD 以及与其它应用程序共享这些 MCD。

图 2-1 描述了由 OS/ 2 多媒体系统提供的 MCD。

图 2-1　媒体控制驱动程序(MCD)的体系结构

2.2　媒体控制驱动程序的入口点

所有的 MCD 都接收以相同的形式传来的命令,而不论它所支持的是流式还是非流式设备。应用程序总是调用 mciSendCommand 或 mciSendString 函数来把命令发向 MCD 的入口点——mciDriverEntry。附带说明一下,有些命令是由 MDM 自己产生的。它们是一些有关保存和恢复一个实例的命令。

表 2-1 给出与 mciDriverEntry 有关的参数。

表 2-1　媒体控制驱动程序的入口点

参　数	说　　明
PVOID pInstance	指向驱动程序的实例数据结构的指针。
USHORT usMessage	请求执行的动作。
ULONG ulParam1	消息的标志。本标志的定义因消息而异。
PVOID pParam2	第二个数据参数,它的意义取决于具体的消息。
USHORT usUserParm	随通知消息返回的用户参数。

mciDriverEntry 的功能是根据不同的消息来切换和执行相应的任务,比如像 MCI_OPEN 这样的消息。

你的驱动程序必须能够按照以下的形式来处理消息。

1 .你的驱动程序必须处理所有传来的消息。

2 .你的驱动程序必须处理它所支持的设备的设备类型消息。例如,如果你在编写一个视盘机的 MCD,你必须分析视盘设备特有消息的特定语法结构。

3 .你的驱动程序必须处理它所支持的设备所特有的消息。假设你的驱动程序控制以下设备类型中的一种:

(1) CD-ROM/ XA 设备

(2) CD 音频设备

(3) 波形音频设备

(4) MIDI 音序器

(5) 数字视频设备

(6) 混响放大设备

(7) 视盘设备

MDM 已经定义好了这些设备的设备类型消息,也就是说,针对以上每一种设备类型列出了一张命令表。如果你想要支持某些设备特有消息,必须创建一张设备特有的命令表。

如果你的驱动程序所要支持的设备类型未在以上列出,你必须创建一张命令表,其中同时包括设备类型消息和设备特有消息。

2 .3　命令消息的类型

媒体设备接口是一套已定义好的并且可扩展的媒体控制命令。MCD 怎样与相应的硬件设备驱动程序通信以执行命令消息,这完全由 MCD 决定。MCD 所使用的设备命令可分为必需命令消息、基本命令消息和系统命令消息。

注意: 关于句法和相关参数,参阅《OS/ 2 多媒体编程参考》。

2 .3 .1　必需命令消息

必需命令消息可被所有设备识别,并对所有媒体设备产生相同的动作。表 2-2 列出

了你的 MCD 必须支持的必需命令消息。

表 2-2 必需命令消息

消 息	说 明
MCI. CLOSE	关闭设备。
MCI. GETDEVCAPS	获取设备的能力。
MCI. INFO	从设备获取文本信息。
MCI. OPEN	初始化一个设备的实例。
MCI. STATUS	从设备获取状态信息。
MCIDRV. SAVE	MDM 向 MCD 发此消息以保存上下文。
MCIDRV. RESTORE	MDM 向 MCD 发此消息,以恢复非活动设备的上下文的状态。

必需命令消息使用一个 ULONG 作为 ulParam1 参数,用以存放本命令消息的标志,并使用 pParam2 参数来指向一个本消息特定的数据结构。你的 MCD 如果想要扩展控制命令,则需要在已有基础上增加新的标志和新的数据结构的字段。当你扩展一个命令消息时,你的 MCD 必须保持对所有必要的标志及数据结构字段的支持。

表 2-3 列出了一些必需命令消息的标志和数据结构。如欲知媒体控制接口命令的详细资料,可参阅《OS/ 2 多媒体编程参考》。

表 2-3 必需命令消息、参数和数据结构

消 息	参数(ulParam1)	数据结构(pParam2)
MCI. CLOSE	MCI. NOTIFY	MCI. GENERIC. PARMS
	MCI. WAIT	
MCI. GETDEVCAPS	MCI. NOTIFY	MCI. GETDEVCAPS. PARMS
	MCI. WAIT	
	MCI. STATUS	
	MCI. GETDEVCAPS. EXTENDED	
	MCI. GETDEVCAPS. MESSAGE	
	MCI. GETDEVCAPS. ITEM	
MCI. INFO	MCI. NOTIFY	MCI. INFO. PARMS
	MCI. WAIT	
	MCI. INFO. PRODUCT	
MCI. OPEN	MCI. WAIT	MMDRV. OPEN. PARMS
	MCI. OPEN. SHARABLE	
	MCI. OPEN. ELEMENT	
	MCI. OPEN. MMIO	
MCI. STATUS	MCI. NOTIFY	MCI. STATUS. PARMS

续表

消息	参数(ulParam1)	数据结构(pParam2)
	MCI_WAIT	
	MCI_STATUS_CLIPBOARD	
	MCI_TRACK	
	MCI_STATUS_ITEM	
MCIDRV_RESTORE	MCI_WAIT	MCI_GENERIC_PARMS
	MCI_SHAREABLE	
	MCI_EXCLUSIVE	
MCIDRV_SAVE	MCI_WAIT	MCI_GENERIC_PARMS

2.3.2 打开一个 MCD

MCI_OPEN 是 MCD 收到的第一条消息。本消息指示驱动程序创建并初始化一个特定设备的实例。MCD 必须为实例的数据结构分配内存并将其初始化。注意,MCI_OPEN 消息并不能激活这个实例。

因为 MDM 要向驱动程序传递附属信息,所以这个为 MCD 打开的结构与结构 MCI_OPEN_PARMS 是不同的。另外,MCD 还需要向 MDM 返回信息。如果应用程序通过 MCI_OPEN 消息请求一个 NOTIFY,那么 MCD 就要通过 MCIDRV_RESTORE 消息返回已打开的 NOTIFY。这对应用程序是透明的。MCD 从 MCI_OPEN 消息中得不到以下的任意一个标志:

- MCI_OPEN_ALIAS
- MCI_NOTIFY
- MCI_OPEN_TYPE_ID

在 MCI_OPEN 消息中,pParam2 指向数据结构 MMDRV_OPEN_PARM,这个结构位于文件 MMDRVOS2.H 中。这个结构包括与 MCD 有关的信息。如表 2-4 示。

表 2-4 MMDRV_OPEN_PARM 结构

字段	输入/输出	说　明
HWND hwndCallback	输入	用于 mciDriverNotify 的窗口句柄。
USHORT usDeviceID	输入	本实例的设备 ID 号。这个字段由 MDM 来填充。
USHORT usDeviceType	输入	本实例的设备类型号。
USHORT usDeviceOrd	输入	本实例的设备序号。
PVOID pInstance	输入输出	指向由驱动程序初始化的实例结构。由驱动程序给此参数赋值。
CHAR szDevDLLName[260]	输入	字符串,包含为此次打开而要调用的设备特定 DLL 的名称。

字段	输入/输出	说　　明
PSZ pszElementName	输入	通常是一个文件名。如果标志置为 OPEN_PLAYLIST,它是一个指向内存播放节目单的指针。如果标志置为 OPEN_MMIO,它是一个 MMIO 句柄。
USHORT usDevParmLen	输入	设备参数数据块的长度。
PVOID pDevParm	输入	设备参数数据块。此数据因不同的设备而异,并可从文件 MMPM2.INI 中查到。(例如, LVD“PORT = COM1 SPEED = 9600N71”)。
USHORT usResourceUnitsRequired	输入输出	本实例所要求的资源单元数,参阅第 2.9 节“资源单元和资源类”。
USHORT usResourceClass	输入输出	本实例所属于的资源类,参阅第 2.9 节“资源单元和资源类”。
USHORT usResourcePriority	输入输出	本实例的资源优无权。
ULONG ulParam2	输入	指向 MCI_OPEN 结构。

2.3.3　子系统消息

MDM 使用下面的消息向 MCD 提供资源管理:

- MCIDRV_SAVE
- MCIDRV_RESTORE

MCIDRV_RESTORE 和 MCIDRV_SAVE 使 MDM 能够告诉 MCD 何时使一个设备的上下文进入活动或非活动状态。这两个命令发出时标志总是置为 MCI_WAIT。尽管 MDM 发出这两条消息,但并不是所有的 MCD 都需要完全地支持它们。MCIDRV_SAVE 和 MCIDRV_RESTORE 消息并不是由应用程序发出的。

在发出 MCI_OPEN 消息之后,MDM 向 MCD 发一条 MCIDRV_RESTORE 消息,以激活设备的上下文。MCD 不应当指望在收到 MCI_OPEN 后立即收到一条 MCIDRV_RESTORE 消息。仅当一个设备上下文已经打开但未激活的情形下,上述情况才会发生。比如,一个设备上下文以非共享方式打开,另一个设备上下文以共享方式打开。MDM 把第二个设备上下文打开并放入非活动堆栈。这样,MCD 不能通过打开来激活设备上下文,而要使用 MCIDRV_RESTORE。

MDM 向 MCD 发一条 MCIDRV_SAVE 消息来使一个实例进入非活动状态。当 MCD 收到这条消息,它要查询设备的状态(它的文件位置、轨道、音量等等)并把这些数据存入该实例的数据结构。此实例此时于是就被置于暂停状态。一但收到恢复命令,驱动程序应当根据保存的实例状态信息重新激活设备。如果在保存时设备正在播放或录制,那么在恢复时它必须被置回这一原来的状态。

以下的命令可以发往非活动的实例:

- MCI_CLOSE

· MCI_INFO

· MCI_STATUS

· MCI_GETDEVCAPS

MCD 应当能够随时为任意的设备上下文处理这些命令。一旦第一次恢复完成,其它命令就可以执行了。

注意: 关于 MCIDRV_RESTORE 和 MCIDRV_SAVE 的详细介绍,参阅《OS/2 多媒体编程参考》一书中的"子系统消息"一章。

2.3.4 等待(Wait)和通知(Notify)标志

MCI_WAIT 表明应用程序此时不能发通知消息,它还表明在当整个命令执行完毕之前,驱动程序不会返回到调用者。

MCI_NOTIFY 表明,一旦执行完一条命令,应当通过 mdmDriverNotify 函数向应用程序发一条通知消息。如果使用了通知(Notify)标志,则 MCD 在返回调用前要执行查错和最快速的处理。查错过程应保证该命令可以启动主处理过程。

如果既没有使用 MCI_WAIT,也没有使用 MCI_NOTIFY,则处理过程与使用了 MCI_NOTIFY 是相同的,只是当命令完成时不通知。

错误和参数检查应当使在控制返回发出调用的线程后发生错误的可能性减到最小。一旦检查到错误,一个通知错误的消息将被发往发出调用的应用程序。查错所依据的错误情况表是随驱动程序的不同而不同的。

命令的主处理过程应当通过一个独立线程来完成,或通过一个可使控制返回发出调用的线程的机制来完成。如果 MCD 使用 SSM 子系统,则在处理通知命令时应使用事件过程。

如果一个 mciDriverEntry 调用返回了一个错误,这个错误不应通过 mdmDriverNotify 发送。在 mciDriverEntry 返回了一个错误之后,MCD 已经完成了对该命令的处理。

接受通知命令后,MCD 有责任保证在返回调用者之前,与命令消息相关的数据结构已被拷贝到驱动程序的内存中。如果没有做到这一点,则驱动程序可能会发现在它可以处理该命令之前这个内存已被更改。例如,MCI_PLAY 消息的 hwndCallback ulFrom 和 ulTo 字段,在使用了 MCI_NOTIFY 标志时就是要拷贝的。

MCI_PLAY,MCI_RECORD,MCI_SEEK 和 MCI_STEP 这些媒体控制消息是要花很多时间来完成的,因为它们是动作命令。而像 MCI_PAUSE,MCI_GETDEVCAPS, MCI_STATUS 和 MCI_SET 这样的非动作命令则不需要很多的处理时间。如果使用了通知标志,非动作命令是不需要用单独的线程来处理的。对于这些命令,应当在命令完成的时候调用 mdmDriverNotify。

如果 MCD 使用流编程接口(Stream Programming Interface,简写为 SPI)函数来传递数据,则要用 SPI 事件来代替独立的线程。SPI 需要一个事件例程来处理 SPI 事件,比如流结尾、错误等等。SPI 将在它的一个线程中调用这个事件例程。于是,mdmDriverNotify 调用应当是这个事件例程的一部分。

一个 MM_MCINOTIFY 消息可以拥有与它相关的下列通知码,如表 2-5 所示。

表 2-5 MM_MCINOTIFY 消息的通知码

通知码	通知的条件
MCI_NOTIFY_SUCCESSFUL	命令成功地完成,无错误。
MCI_NOTIFY_SUPERSEDED	在设置了通知标志的情况下接到第二个同类型的命令。
MCI_NOTIFY_ABORTED	接到了第二个命令,以致第一个命令不能成功地完成。若驱动程序在参数检查或命令处理过程中发现一个错误,错误要被返回给调用者,没有通知返给应用程序。

如果 MM_MCINOTIFY 通知码字段包含的值不在表 2-5 中,则该值是某特定的 OS/2多媒体错情况的错误码。对于每一个命令只能发一条 MM_MCINOTIFY 消息。

2.3.5 基本命令消息

基本命令是指这样的一类命令,所有的设备类型都理解它,但能够修改它的参数。比如,当向视盘机发出一个 PLAY 命令时,你尽可以指定以帧每秒为单位的播放速度。然而对于像 CD 机一类的设备,它可能就不具有按不同速度播放的功能。

基本命令的清单如表 2-6 所示。如果一个设备不使用某个设备类型命令,它会返回 MCIERR_UNSUPPORTED_FUNCTION。如果一个设备支持该命令,但不是全部的选项,它会对所不适用的选项返回 MCIERR_UNSUPPORTED_FLAG。

表 2-6 设备类型的基本命令消息

消 息	说 明
MCI_CONNECTOR	允许、禁止、计数或查询连接符的状态。
MCI_LOAD	将新的设备元素(文件名)加入已打开的设备上下文。
MCI_MASTERAUDIO	为系统中所有的音频设备设定主音频设置。MCI_MASTERAUDIO 也被用作系统命令,当驱动程序第一次打开时用于查询当前的音频设置。
MCI_PAUSE	挂起设备的播放。
MCI_PLAY	启动设备的播放。
MCI_RECORD	启动录制数据。
MCI_RESUME	从暂停状态恢复播放或录制。
MCI_SAVE	保存设备的数据。
MCI_SEEK	转移到指定的位置然后停止。
MCI_SET	设定设备的操作状态。
MCI_SETCUEPOINT	设定提示点。
MCI_SETPOSITIONADVISE	为设备设定位置改变通知。
MCI_STATUS	获取关于媒体设备的状态的信息。
MCI_STOP	停止设备。

基本命令消息使用 ulParam1 参数表示那些可用于本命令消息的标志。它也用 pParam2 参数来指向由消息特定的数据结构。你的 MCD 可以增加标志及参数以创造扩展的

命令。当你扩展一个命令消息时,你的 MCD 必须仍然能够响应基本标志和参数。

表 2-7 列出了基本命令消息的标志和数据结构。关于媒体控制接口命令的详细资料,可参阅《OS/ 2 多媒体编程参考》一书。

表 2-7　基本命令消息、参数和数据结构

消　　息	参数(ulParam1)	数据结构(pParam2)
MCI. CONNECTOR	MCI. NOTIFY	MCI. CONNECTOR. PARMS
	MCI. WAIT	
	MCI. ENABLE. CONNECTOR	
	MCI. DISABLE. CONNECTOR	
	MCI. QUERY. CONNECTOR. STATUS	
	MCI. CONNECTOR. TYPE	
	MCI. CONNECTOR. INDEX	
MCI. LOAD	MCI. OPEN. ELEMENT	MCI. LOAD. PARMS
	MCI. OPEN. MMIO	
	MCI. NOTIFY	
	MCI. WAIT	
MCI. MASTERAUDIO	MCI. WAIT	MCI. MASTERAUDIO. PARMS
	MCI. QUERYCURRENTSETTING	
	MCI. QUERYSAVEDSETTING	
	MCI. SAVESETTING	
	MCI. MASTERVOL	
	MCI. SPEAKERS	
	MCI. HEADPHONES	
	MCI. ON	
	MCI. OFF	
MCI. PAUSE	MCI. NOTIFY	MCI. GENERIC. PARMS
	MCI. WAIT	
MCI. PLAY	MCI. NOTIFY	MCI. PLAY. PARMS
	MCI. WAIT	
	MCI. FROM	
	MCI. TO	
MCI. RECORD	MCI. NOTIFY	MCI. RECORD. PARMS
	MCI. WAIT	
	MCI. FROM	
	MCI. TO	
	MCI. RECORD. INSERT	
	MCI. RECORD. OVERWRITE	
MCI. RESUME	MCI. NOTIFY	MCI. GENERIC. PARMS
	MCI. WAIT	
MCI. SAVE	MCI. NOTIFY	MCI. SAVE. PARMS

续表

消　　息	参数(ulParam1)	数据结构(pParam2)
	MCI. WAIT	
	MCI. SAVE. FILE	
MCI. SEEK	MCI. NOTIFY	MCI. SEEK. PARMS
	MCI. WAIT	
	MCI. TO	
	MCI. TO. START	
	MCI. TO. END	
MCI. SET	MCI. NOTIFY	MCI. SET. PARMS
	MCI. WAIT	
	MCI. SET. AUDIO	
	MCI. SET. DOOR. OPEN	
	MCI. SET. DOOR. CLOSED	
	MCI. SET. DOOR. LOCK	
	MCI. SET. DOOR. UNLOCK	
	MCI. SET. VOLUME	
	MCI. OVER	
	MCI. SET. VIDEO	
	MCI. SET. ON	
	MCI. SET. OFF	
	MCI. SET. SPEED. FORMAT	
	MCI. SET. TIME. FORMAT	
	MCI. SET. ITEM	
MCI. SETCUE-POINT	MCI. NOTIFY	MCI. CUEPOINT. PARMS
	MCI. WAIT	
	MCI. SET. CUEPOINT. ON	
	MCI. SET. CUEPOINT. OFF	
MCI. SETPOSI-TIONADVISE	MCI. NOTIFY	MCI. POSITION. PARMS
	MCI. WAIT	
	MCI. SET. POSITION. ADVISE. ON	
	MCI. SET. POSITION. ADVISE. OFF	
MCI. STATUS	MCI. NOTIFY	MCI. STATUS. PARMS
	MCI. WAIT	
	MCI. STATUS. START	
	MCI. TRACK	
	MCI. STATUS. ITEM	
MCI. STOP	MCI. NOTIFY	MCI. GENERIC. PARMS
	MCI. WAIT	

2.3.6 系统命令消息

系统命令直接由媒体设备管理器(MDM)理解,并不发往 MCD。表 2-8 列出了 MDM 支持的系统命令。

表 2-8 系统命令

命 令	说 明
MCI. ACQUIREDEVICE	获取对设备物理资源的使用。
MCI. CONNECTION	返回关于设备上下文的连接的信息。
MCI. CONNECTORINFO	返回关于设备连接符的信息。
MCI. DEFAULTCONNECTION	建立、打断或查询一个默认的连接。
MCI. GROUP	提供用单一命令对多个设备的控制。
MCI. MASTERAUDIO	当驱动程序第一次打开时,查询当前的音频设置。此命令用作基本命令用于以后调节主音频。
MCI. SYSINFO	获取系统中所安装的设备的信息。
MCI. RELEASEDEVICE	释放对设备的物理资源的独占使用。

2.3.7 命令处理

所有的 MCD 都接收以相同的形式传来的命令,而不论它所支持的是流式还是非流式设备。应用程序调用 mciSendCommand 或 mciSendString 函数来把命令发向 MCD 的入口点——mciDriverEntry。另外,有些命令是由 MDM 自己产生的。它们是有关保存和恢复一个实例和同步化消息的命令。同步化消息是指在一个设备组中,由主设备定时地产生并发向从设备的消息。

尽管流式和非流式设备接收以相同形式传来的命令,但它们却以不同的方式处理这些来自 MDM 的命令。这两者的区别将在以后的章节中谈到,我们在这里将其总结为表 2-9。

表 2-9 MCD 命令处理的比较

当 MDM 发送	波形音频 MCD	CD 音频 MCD
MCI. SET 消息以设置音频属性	将消息送往混响放大器。	使用 IOCtl 来处理消息。
控制播放的消息(如, SEEK, PLAY, PAUSE, STOP)	将消息送往 SSM。	使用 IOCtl 来处理消息。

2.3.8 错误返回码

错码返回码应当在所有的 MCD 中保持一致这样才会使 MCD 在应用程序看来具有一致性。表 2-10 是对使用普通的错误返回码的指导。

表 2-10　对使用普通的错误返回码的指导

当发生如下错误时	使用如下错误消息
所给的值超出范围	MCIERR. OUTOFRANGE
不明的 usMessage 值	MCIERR. UNRECOGNIZED. COMMAND
不明的 ulParam1 值	MCIERR. INVALID. FLAG
ulItem 字段中的不明标志	MCIERR. INVALID. ITEM. FLAG
ulTimeFormat 字段中的不明标志	MCIERR. INVALID. TIME. FORMAT. FLAG
ulSpeedFormat 字段中的不明标志	MCIERR. SPEED. FORMAT. FLAG
无效的或空(NULL)的 pParam2 字段	MCIERR. MISSING. PARAMETER
pParam2 中无效的输出缓冲区地址	MCIERR. INVLID. BUFFER
当要求有一个或一个以上的标志时,ulParam1 中缺少标志	MCIERR. MISSLNG. FLAG
ulParam1 中的标志对消息而言是有效的,但驱动程序无法执行该任务	MCIERR. UNSUPPORTED. FLAG
usMessage 中所要求的功能,虽为设备所具有,但驱动程序不支持	MCIERR. UNSUPPORTED. FUNCTION
设置了多于一个的双向独占标志	MCIERR. FLAGS. NOT. COMPATIBLE

以下的错误总是由 MDM 处理的:

· 当设置了 MCI. NOTIFY 标志时,为消息说明了一个无效的回调句柄。

· 同时使用了 MCI. WAIT 和 MCI. NOTIFY 标志。

· 向一个非活动的实例发命令。

2.3.9　返回值和返回类型

MCD 应当尽可能地使用 MCIERR 返回码。这些返回码由 C 头文件 MEERROR.H 给出。源出于 SPI,MMIO 或操作系统的错误信息码都应当转换归结为 MCIERR 返回码,只要保证这么做是有意义的。如果驱动程序在某条件下需要独特的返回码,它可以使用从 MCIERR. CUSTOM. DRIVER. BASE 到 MEBASE. 1 的错误号。如果要使用用户自定义的驱动程序返回码,必须创建一张用户自定义驱动程序错误表。

当媒体消息有一个返回字段时,MCD 应在返回码的高位字中使用表 2-11 的返回类型。

表 2-11　返回类型

返回类型	意　　义
MCI. INTEGER. RETURENED	将给定的二进制数转换为它的无符号 ASCII 字符串形式。
MCI. COLONIZED2. RETURN	将二进制数转换为以下形式: byte 0:byte 2
MCI. COLONIZED3. RETURN	将二进制数转换为以下形式: byte 0∶byte 1∶byte 2
MCI. COLONIZED4. RETURN	将二进制数转换为以下形式:

续表

返回类型	意　　义
	byte 0∶byte 1∶byte 2∶byte 4
MCI.TRUE.FALSE.RETURN	将 TURE/ FALSE 值转换为字符串形式。
MCI.DEVICENAME.RETURN	将设备类型号转换为它的设备名字符串。
MCI.SPEED.FORMAT.RETURN	将速度格式整数转换为相应字符串。
MCI.TIME.FORMAT.RETURN	将时间格式整数转换为相应字符串。
MCI.MEDIA.TYPE.RETURN	将视盘类型号转换为“CLV”,“CAV”,“OTHER”。
MCI.TRACK.TYPE.RETURN	将轨道类型号转换为“AUDIO”,“DATA”,“OTHER”。
MCI.CONNECTOR.TYPE.RETURN	将连接符类型号转换为相应的字符串。
MCI.CDXA.CHANNEL.DESTINATION.RETURN	将 CDXA 的声道目标类型号转换为相应的字符串。
MCI.PREROLL.TYPE.RETURN	将预处理类型号转换为相应的字符串。
MCI.FORMAT.TAG.RETURN	将格式标记(tag)类型号转换为相应的字符串。
MCI.SIGNED.INTERER.RETURN	将给定的二进制数转换为它的有符号 ASCII 字符串形式。

表 2-11 中列出的返回类型使句法分析程序可以把二进制数转换为字符串。这种转换仅当应用程序调用了 mciSendString 并且在调用过程中无错误发生的情况下,才能进行。例如:“capability cdaudio has video”将使 mciSendString 的返回字符串字段返回 FALSE。CD 音频 MCD 的媒体驱动程序在 MCI.GETDEVCAPS.PARMS 结构的 ulReturn字段写入零,并把 MCI.TRUE.FALSE.RETURN 写入返回码的高位字。

2.4　增加新的命令消息

媒体驱动程序的编写者应当按照以下 4 步来增加驱动程序对设备特有消息的支持:

1.定义你的驱动程序所要支持的新增的或经过修改的媒体控制消息。(参阅 2.4.1 节“定义新的消息”。)

2.为这个消息定义新的数据结构和标志。(参阅 2.4.2 节“定义新的数据结构和标志”。)

3.创建一张用户自定义命令表。这样,MDM 就能利用这张表对 mciSendString 中的命令字符串进行句法分析,然后创建相应的数据结构并把它传递给你的驱动程序。(参阅 2.5.2“节命令清单的句法分析”。)

4.在你的 MCD 的入口点(mciDriverEntry)支持这条新的消息。

2.4.1　定义新的消息

假设你试图为一个支持视盘机的 MCD 定义一个新的复位(reset)命令,你可以通过

发送字符串 reset program_number 来调用这条命令。这条命令将把视盘机的所有参数设为默认值,或由 program_number 指定的数种预定义状态之一。

你要在 mciDriverEntry 所支持的消息的清单中加入以下内容:

```
#define MCI_VD_RESET   MCI_USER_MESSAGES
```

MCI_USER_MESSAGE 是你能用作用户自定义消息的第一个整数。鉴于资源编译器不能接受 RCDATA 资源类型中的数学表达式,你在这里必须指定所要使用的具体数值。

至此,你已经定义好了这个消息的 ID 号。下面需要做的就是定义数据结构和命令表,并在 mciDriverEntry 中创建一个代码作为这条消息的句柄。

2.4.2 定义新的数据结构和标志

大多数的新增消息需要附加参数,以说明它的确切功能。媒体控制接口消息使用 pParam 2 作为指向数据结构的指针,并使用 ulParam 1 作为位字段来表示与消息相关的标志。在从发出调用的应用程序接受数据的数据结构中,每一个字段都有一个标志。应用程序使用标志来设定 ulParam1 的位字段,以指示把数值赋给一个特定的字段,标志的使用是无参数的,这样标志就无需与数据结构中的某个字段一一对应。

2.4.2.1 定义新的数据结构

一个媒体控制接口消息的数据结构的字段,其长度总是与 ULONG 相同的。数据结构中具体的字段数目取决于具体的消息。其中的第一个字段必须保留,作为与 MCI_NOTIFY 标志同时使用的窗口函数的句柄。而以后的各字段则由消息返回的数据的类型决定。

· 如果不返回数据,则在数据结构中无需保留返回字段。向 MCD 传递信息的数据字段直接跟在 hwndCallback 字段的后面。例如,图 2-2 所示的 MCI_SAVE_PARMS 结构中,就没有返回字段。

```
typedef struct_MCI_SAVE_PARMS {
    HWND    hwndCallback;   /* PM window handle for MCI notify message */
    PSZ     pszFileName;    /* File name to save data to               */
 } MCI_SAVE_PARMS;
typedef MCI_SAVE_PARMS   *PMCI_SAVE_PARMS;
```

图 2-2 MCI_SAVE_PARMS 的数据结构

· 如果返回的是整数型数据,则数据结构的第二个字段保留给所返回的数据。所有向 MCD 传递信息的字段从第三个字段开始。例如,图 2-3 所示的 MCI_GETDEVCAPS_PARMS 结构,它使用 ulReturn 作为整数型返回字段。

```
typedef struct_MCI_GETDEVCAPS_PARMS {
    HWND     hwndCallback;  /* PM window handle for MCI notify message */
    ULONG    ulReturn;      /* Return field                             */
    ULONG    ulItem;        /* Item field for GETDEVCAPS item to query */
    USHORT   usMessage;     /* Field to hold MCI message to query       */
    USHORT   usReserved0;   /* Reserved field                           */
 } MCI_GETDEVCAPS_PARMS;
typedef MCI_GETDEVCAPS_PARMS   *PMCI_GETDEVCAPS_PARMS;
```

图 2-3　MCI_GETDEVCAPS_PARMS 数据结构

· 如果返回的是字符串型数据,则第二和第三个字段都保留给返回的数据。其中第二个 ULONG 字段保存一个指向以 null 结尾的返回字符串的指针。第三个 ULONG 字段则保存返回缓冲区的长度。这个返回字符串的缓冲区要由应用程序负责创建。所有向 MCD 传递信息的字段从第四个字段开始。例如,图 2-4 所示的 MCI_INFO_PARMS 结构,它使用 pszReturn 和 ulRetSize 作为返回字段。

```
typedef struct_MCI_INFO_PARMS {
    HWND       hwndCallback;  /* PM window handle for MCI notify message */
    PSZ        pszReturn;     /* Pointer to return buffer                 */
    ULONG      ulRetSize;     /* Return buffer size                       */
 } MCI_INFO_PARMS;
typedef MCI_INFO_PARMS   *PMCI_INFO_PARMS;
```

图 2-4　MCI_INFO_PARMS 数据结构

· 如果返回的是 RECTL 型数据,则数据结构的第二到第五个字段都保留给返回数据。第一个 ULONG 位置保留给 RECTL 型数据的左值,第二个 ULONG 位置保留给 RECTL 型数据的底值。所有向 MCD 传递信息的字段从第六个 ULONG 位置开始。实际上,多数的数据结构定义中,并不为 RECTL 型数据使用两个 ULONG,而是使用一个 RECTL 字段来代替。例如,图 2-5 所示的 MCI_VID_RECT_PARMS 结构,它使用 rc 作为返回字段。

```
typedef struct_MCI_VID_RECT_PARMS {
    HWND     hwndCallback;  /* PM window handle for MCI notify message */
    RECTL    rc;            /* rectangle array specifying the offset   */
                            /* and size of a rectangle                 */
} MCI_VID_RECT_PARMS;
```

图 2-5　MCI_VID_RECT_PARMS 数据结构

2.4.2.2　赋标志值

除了指定说明一个数据结构中所要用的字段,标志还可以无参数地指定一个选项。

例如,MCI_WAIT 标志就不使用任何参数。

当你增加一个新的标志时,你必须确信它不会与已经有的标志相冲突。在 ULONG 的 32 位中,第 0 位到第 15 位都是为 MDM 保留的。第 16 位(0x00010000)是驱动程序可以供它的标志使用的第一位。如果你的命令消息是一个新增的命令消息,你所选择的位不能与已为该命令定义好的标志相冲突。在新标志中任何未使用的位都必须置为 0。例如,一个视盘机的新增命令使用了标志 16 至 20,那么对这个新命令的用户自定义扩展则可以使用标志 21 至 31。

让我们继续那个为视盘机增加一条 RESET 命令的例子。图 2-6 所示的代码样例中定义了一个数据结构,一个相应的指针,及一个与程序中数据结构字段相对应的标志。

```
typedef struct {
    HWND    hwndCallback;
    ULONG   ulProgram;
} MCI_VD_RESET_PARMS;

typedef MCI_VD_RESET_PARMS FAR * PMCI_VD_RESET_PARMS;

#define MCI_VD_RESET_PROGRAM   0x00010000L
```

图 2-6　赋标志值

当应用程序在 ulParam1 中设定了 MCI_VD_RESET_PROGRAM 标志,这意味着一个数值要赋给字段 ulProgram。

至此,你已经创建了消息命令。数据结构和标志,接下去要做的是创建一张命令表,以告诉 MDM 怎样把以字符串形式给出的命令翻译成为消息命令的格式。参阅 2.5.2 节“命令清单的句法分析”。

2.5　命　令　表

命令表是这样一个结构,它使 MDM 的字符串分析程序可以把命令字符串翻译给 MCD。这就为你的 MCD 提供了字符串命令接口的支持。对驱动程序来说,命令表是一种资源,它是用 RCDATA 资源类型创建的。RCDATA 块的资源号就是设备类型号。

图 2-7 表示了媒体设备管理器(MDM)怎样使用命令表,一个可运行程序或 mciRxInit 命令通过 mciSendString 函数把一条字符串命令传给 MDM。然后,MDM 调用它的字符串句法分析程序翻译这条字符串命令,将其转换为相应的过程化的命令。MDM 按照从最特别到最一般的顺序扫描命令表。它首先扫描用户自定义的设备特有的表(如果 MCD 中存在这样的表的话)。然后,MDM 再依次在设备类型表和系统默认表中搜索这个字符串。

如图 2-7 所示,MDM 提供了设备类型表,以支持 OS/2 的多媒体逻辑设备。如果你不需要增加或修改命令,那么这些“内装”的表就将满足你的 MCD 的要求。如果你的 MCD 有一些新要求(比如为一条命令增加新的标志或更改其数据结构),你必须创建一张

图 2-7　命令表

用户自定义表以定义你的设备所特有的命令。

用户自定义表使你可以通过新增或更改命令入口的方式来创建你自己的版本的一套媒体控制接口命令。例如,数字视频的设备类型表支持 SEEK **标志,如图** 2-8 **所示。**

```
"seek",          MCI_SEEK,0,        MCI_COMMAND_HEAD,
   "notify",     MCI_NOTIFY,        MCI_FLAG,
   "wait",       MCI_WAIT,          MCI_FLAG,
   "to start",   MCI_TO_START,      MCI_FLAG,
   "to end",     MCI_TO_END,        MCI_FLAG,
   "to",         MCI_TO,            MCI_INTEGER,
"",              0L,                MCI_END_COMMAND,
```

图 2-8　数字视频设备类型的 SEEK 命令集

然而,你却希望你的 MCD **能查找最近的位置。要达到此目的,你可以在你的** MCD **中创建一张用户自定义表,以便为数字视频播放机进行命令句法分析(如图** 2-9 **所示)。注意,用户自定义表必须包括与所修改的命令相关的所有参数,而不仅仅是新增的或更改的参数。**

```
"seek",          MCI_SEEK,0,        MCI_COMMAND_HEAD,
   "notify",     MCI_NOTIFY,        MCI_FLAG,
   "wait",       MCI_WAIT,          MCI_FLAG,
   "to start",   MCI_TO_START,      MCI_FLAG,
   "to nearest", MCI_TO_NEAREST,    MCI_FLAG,
```

```
    "to end",          MCI_TO_END,          MCI_FLAG,
    "to",              MCI_TO,              MCI_INTEGER,
  "",                  0L,                  MCI_END_COMMAND,
```

图 2-9 用户自定义命令表的 SEEK 命令表

在编写 MCD 时, 所有 DLL 文件的名字, (其中包括了命令表资源) 必须注册在 MMPM2.INI 文件。当安装你的 MCD 时, MDM 将根据 MCD 命令表中的信息来刷新这个 INI 文件。详见第 5 章 5.5 节“安装媒体控制驱动程序”。

图2-10 的例子显示了命令表入口以怎样的形式出现在 MMPM2.INI 文件中。MCDTABLE 一项包括了 MDM.DLL 文件, 而设备类型表和系统的默认表就包含于这个文件中。VSDTABLE 一项则表示有一个用户自定义命令表资源存在于文件 SVMC.DLL 中。

MDM 按照从最特别到最一般的顺序扫描命令表。在图 2-10 所示的例了中, MDM 首先扫描 SVMC.DLL 文件中的用户自定义表资源, 然后再到 MDM.DLL 文件里依次查询系统内置的数字视频设备类型表(SMVCMD.RC)及系统默认表(MDMCMD.RC)。

```
[ibmdigvid 01]
  VERSIONNUMBER = 1
  PRODUCTINFO = SOFTWARE MOTION VIDEO
  MCDDRIVER = SVMC
  VSDDRIVER = AUDIOIF
  PDDNAME = AUDIO1$
  MCDTABLE = MDM
  VSDTABLE = SVMC
  RESOURCENAME = DIGITALVIDEO
  DEVICEFLAG = 1
  DEVICETYPE = 12
  SHARETYPE = 3
  RESOURCEUNITS = 1
  RESOURCECLASSES = 1,1
  VALIDCOMBINATIONS =
  EXTNAMES = 3,UMB,MMM,V
  ALIASNAME = DV
```

图 2-10 在 MMPM2.INI 中引用命令表

2.5.1 命令表的句法

命令表由命令清单组成。表中每一个命令的清单定义了特定命令的句法分析结构。例如, 图 2-11 是 MCI_OPEN 和 MCI_INFO 的命令清单。这是 MDMCMD.RC 文件中的

系统默认表的一部分。

```
"open",          MCI_OPEN,0,          MCI_COMMAND_HEAD,
   "",           MCI_INTEGER,         MCI_RETURN,
   "notify",     MCI_NOTIFY,          MCI_FLAG,
   "wait",       MCI_WAIT,            MCI_FLAG,
   "readonly",   MCI_READONLY,        MCI_FLAG,
   "shareable",  MCI_OPEN_SHAREABLE,  MCI_FLAG,
   "type",       0L,                  MCI_STRING,
   "",           MCI_OPEN_ELEMENT,    MCI_STRING,
   "alias",      MCI_OPEN_ALIAS,      MCI_STRING,
"",              0L,                  MCI_END_COMMAND,

"info",          MCI_INFO,0,          MCI_COMMAND_HEAD,
   "",           MCI_STRING,          MCI_RETURN,
   "notify",     MCI_NOTIFY,          MCI_FLAG,
   "wait",       MCI_WAIT,            MCI_FLAG,
   "product",    MCI_INFO_PRODUCT,    MCI_FLAG,
   "file",       MCI_INFO_FILE,       MCI_FLAG,
"",              0L,                  MCI_END_COMMAND,
```

图 2-11　MDMCMD.RC 中的 MCI_OPEN 和 MCI_INFO 命令清单

命令清单依顺序写成三列:

· 第 1 列是命令及它的标志,为命令字符串格式。

· 第 2 列是命令及它的标志,为命令消息格式。

· 第 3 列是命令清单中每一行的行类型。

2.5.1.1　行类型

MDM 的字符串句法分析程序通过行类型来确定怎样翻译每一行。一个命令行以行类型 MCI_COMMAND_HEAD 开始,以行类型 MCI_END_COMMAND 结束。

1. MCI_RETURN

这个行类型表示本命令提供了返回信息。MCI_RETURN 必须在命令清单的第二行。

· 第 1 列为"　"。

· 第 2 列为返回类型:整数或字符串。

· 第 3 列为 MCI_RETURN。

一个命令若返回整数,则该返回值必为一个 ULONG。若返回字符串,则需要两个 ULONG:第一个 ULONG 保存返回缓冲区的指针,第二个 ULONG 保存返回缓冲区的长

度。

句法分析程序利用 mciSendString 函数的 pszReturnString 和 usReturnLength 两个参数发回返回信息。图 2-11 显示了两种返回类型的使用。

MCI_OPEN 返回一个 ULONG 值。MCI_INFO 的返回数据填入了一个缓冲区,而整个命令的结构中的第二个 ULONG 里的指针正指向这个缓冲区。(记住,第一个 ULONG 永远是 hwndCallback。)

2 . MCI_RETURN_TYPE

字符串句法分析程序根据这个行类型把返回值转换为字符串,如图 2-12。

```
"",           MCI_TRUE_FALSE_RETURN,0,    MCI_RETURN_TYPE,
  "TRUE",     1L,                         MCI_RETURN_TYPE_STRING,
  "FALSE",    0L,                         MCI_RETURN_TYPE_STRING,
"",           0L,                         MCI_END_RETURN_TYPE,
```

图 2-12 MCI_RETURN_TYPE 项

返回行的标志定义字段将与返回码的高位字相比较。如果两者匹配,则语句分析程序将再在标志定义字段与返回值之间寻求匹配。如果与返回值匹配成功,则关键词字符串将通过对 mciSendString 函数的调用,以 pszReturnString 字段返回到应用程序。

3 . MCI_FLAG

这个行类型用以把关键词转换为标志值并赋给 ulParam1。如果在分析命令字符串的过程中遇到了一个以上的 MCI_FLAG 行类型,则参数 ulParam1 是关键词清单的标志定义字段相或的结果。

4 . MCI_STRING

这个行类型用于消息的字符串的值。跟在 MCI_STRING 关键词之后的字符串将被拷贝到命令结构中去。MDM 分配内存来存放这个字符串,并把它的地址放入命令结构。

5 . MCI_INTEGER

这个行类型用于消息的整数的值。跟在 MCI_INTEGER 关键词之后的整数将被拷贝到命令结构中去。这个整数的值一个由字符串形式转换而来的 ULONG。

6 . MCI_CONSTANT

这个行类型与 MCI_END_CONSTANT 行类型一起,允许用一组可能的关键词来代表一个标志或一个整数。在图 2-13 中,任一个时间格式值都可作为时间格式常数(time format constant)的值。

```
"",                               MCI_SET_TIME_FORMAT,         MCI_CONSTANT,
  "time format milliseconds",     MCI_FORMAT_MILLISECONDS,     MCI_INTEGER,
  "time format ms",               MCI_FORAMT_MILLISECONDS,     MCI_INTEGER,
  "time format mmtime",           MCI_FORAMT_MMTIME,           MCI_INTEGER,
"",                               0L,                          MCI_END_CONSTANT,
```

图 2-13　MCI_CONSTANT 项

MCI_CONSTANT 行类型对应的值是与 ulParam1 字段相或的结果。在这个常数块中与 MCI_INTEGER 行类型对应的值将被拷贝到命令结构中去。这个常数块在命令结构中只是一个 ULONG。

7．MCI_DEFAULT_STRING

这个行类型与 MCI_DEFAULT_INTEGER 行类型,可使一个不确定类型的值用作一个字符串或一个整数。例如,LOAD 命令(图 2-14 所示)使用一个默认的字符串型来放文件名字符串。

```
"load",            MCI_LOAD,0,          MCI_COMMAND_HEAD,
   "notify",       MCI_NOTIFY,          MCI_FLAG,
   "wait",         MCI_WAIT,            MCI_FLAG,
   "new",          0L,                  MCI_FLAG,
   "readonly",     MCI_READONLY,        MCI_FLAG,
   "",             MCI_OPEN_ELEMENT,    MCI_DEFAULT_STRING,
"",                0L,                  MCI_END_COMMAND,
```

图 2-14　MCI_DEFAULT_STRING 项

8．MCI_RECTL

这个行类型表明 RECTL 型数据要修改本项对应的媒体控制接口标志。本项的第二列的内容就是要设定的标志。消息命令的数据结构必须保留一个 ULONG 来存放该整数值。

9．MCI_OR

这个行类型使你可以在同一个结构中混合使用不同的行类型。图 2-15 显示了一个 MCI_OR 项的例子,这个例子来自 MDMCMD.RC 文件中的 MCI_STATUS 命令清单。注意,在命令清单中可使用%d 符号来指示打印一个十进制整数值。

```
"",            0L,                          MCI_OR,
"track"        MCI_TRACK,                   MCI_INTEGER,
"channel",     0L,                          MCI_CONSTANT,
    "all",     MCI_STATUS_AUDIO_ALL,        MCI_INTEGER,
    "left",    MCI_STATUS_AUDIO_LEFT,       MCI_INTEGER,
    "right",   MCI_STATUS_AUDIO_RIGHT,      MCI_INTEGER,
    "%d",      0L,                          MCI_INTEGER,
 "",           0L,                          MCI_END_CONSTANT,
 "",           0L,                          MCI_END_OR,
```

图 2-15　MCI_OR 项

10．MCI_STRING_LIST

指向清单中一个字符串指针数组的指针。例如,图 2-16 中用 MCI_STRING_LIST 来指向一个设备标识符的指针数组。

```
"group",            MCI_GROUP,0,          MCI_COMMAND_HEAD,
   "delete",        MCI_GROUP_DELETE,     MCI_FLAG,
   "nopiecemeal",   MCI_NOPIECEMEAL,      MCI_FLAG,
   "synchronize",   MCI_SYNCHRONIZE,      MCI_FLAG,
   "wait",          MCI_WAIT,             MCI_FLAG,
   "notify",        MCI_NOTIFY,           MCI_FLAG,
   "",              0L,                   MCI_INTEGER,
   "",              0L,                   MCI_INTEGER,
   "master",        MCI_GROUP_MASTER,     MCI_STRING,
   "",              0L,                   MCI_STRING,
   "",              0L,                   MCI_INTEGER,
   "make",          MCI_GROUP_MAKE,       MCI_STRING_LIST,
"",                 0L,                   MCI_END_COMMAND,
```

图 2-16 MCI_STRING_LIST 项

2.5.2 命令清单的句法分析

一个命令表由命令清单组成,后者告诉 MDM 的句法分析程序如何分析一条命令。例如,图 2-17 给出了一个 MCI_SEEK 的命令清单。

```
"seek",          MCI_SEEK,0,        MCI_COMMAND_HEAD,
   "notify",     MCI_NOTIFY,        MCI_FLAG,
   "wait",       MCI_WAIT,          MCI_FLAG,
   "to start",   MCI_TO_START,      MCI_FLAG,
   "to end",     MCI_TO_END,        MCI_FLAG,
   "to",         MCI_TO,            MCI_INTEGER,
"",              0L,                MCI_END_COMMAND,
```

图 2-17 命令清单

当句法分析程序发现 mciSendString 的 pszCommand 参数为"seek"时,由 SEEK 命令清单来告诉它怎样创建相应的数据结构,以使 mciSendString 的 pParam2 参数指向这个结构。

注意:每一行被分解成一个以 null 结尾的字符串和两个 ULONG。跟在"seek"之后的那个 ULONG 由两部分组成:MCI_SEEK 和 0。命令消息(在这里,就是 MCI_SEEK)是一个 USHORT 型,故而我们需要再一个 USHORT 来补满一个 ULONG。MCI_COMMAND_HEAD 行类型告诉 MDM,第一个 ULONG(实际上是第一个 USHORT)包含的是 mciDriverEntry 的 usMessage 参数。

如果某行的行类型是 MCI_FLAG,那么本行的第一个 ULONG 要与所有其它具有 MCI_FLAG 行类型的行中的 ULONG 进行或运算,以此来构造出 ulParam1 参数。

字符串句法分析程序分两次处理命令清单。第一次,决定要为命令分配的结构的长度;第二次,分析命令字符串,填充命令结构,并创建一个 ulParam1 标志。

每条命令由若干 ULONG 组成。所有的命令至少含有一个 ULONG,以作为 hwndCallback 字段。句法分析程序自动提供这个 ULONG。命令结构其余部分的大小取决于该命令所用的行类型。

如果某行的行类型是 MCI_INTEGER,那么跟在字符串 S 之后的文本(这里 S 是指本命令行中的 null 结尾的字符串),作为 pszCommand 参数,是一个整数。它将被放入由 pParam2 所指向的那个由若干 ULONG 组成的结构的相应字段。

如果某行的行类型是 MCI_STRING,那么跟在字符串 S 之后的文本,作为 pszCommand 参数,是一个字符串。这个字符串的指针将被放入由 pParam2 所指向的那个由若干 ULONG 组成的结构的相应字段。

最后,如果行类型是 MCI_RETURN,那么第一个 ULONG 就是要返回的值。它将通过由 pParam2 所指向的数据结构返回。

pParam2 指向的数据结构是根据命令清单生成的。它的第一个 ULONG 永远是 hwndCallback。紧接下去的一个或两个 ULONG 则代表 MCI_RETURN:

- 在 MCI_RETURN 这一行,如果第一个 ULONG 是 MCI_INTEGER,那么 pParam2 的第二个 ULONG 就是 ulReturn。
- 在 MCI_RETURN 这一行,如果第一个 ULONG 是 MCI_STRING,那么 pParam2 的第二个 ULONG 就是 ulReturn,第三个 ULONG 就是 ulRetSize。ulRetSize 给出了字符串的长度。

pParam2 的其它字段将根据在命令行的第二个 ULONG 中出现的 MCI_INTEGER 和 MCI_STRING 一一填入。另外,MCI_CONTANT 和 MCI_END_CONTANT 构造了一个结构中某字段可取值的范围。这个常数块在结构中只占一个 ULONG。这些字段将根据它们在命令清单中出现的次序而一一填入。例如,假设字符串句法分析程序分析如图 2-18 所示的 MCI_STATUS 命令清单。

```
"status\0",              MCI_STATUS,0,                  MCI_COMMAND_HEAD,
"\0",                    MCI_INTEGER,                   MCI_RETURN,

"notify\0",              MCI_NOTIFY,                    MCI_FLAG,
"wait\0",                MCI_WAIT,                      MCI_FLAG,
"\0",                    MCI_STATUS_ITEM,               MCI_CONSTANT,
"mode\0",                MCI_STATUS_MODE,               MCI_INTEGER,
"ready\0",               MCI_STATUS_READY,              MCI_INTEGER,
"current track\0",       MCI_STATUS_CURRENT_TRACK,      MCI_INTEGER,
"length\0",              MCI_STATUS_LENGTH,             MCI_INTEGER,
"number of tracks\0",    MCI_STATUS_NUMBER_OF_TRACKS,   MCI_INTEGER,
"position\0",            MCI_STATUS_POSITION,           MCI_INTEGER,
```

```
"position in track \0",      MCI_STATUS_POSITION_IN_TRACK,  MCI_INTEGER,
"time format \0",            MCI_STATUS_TIME_FORMAT,        MCI_INTEGER,
"speed format \0",           MCI_STATUS_SPEED_FORMAT,       MCI_INTEGER,
"\0",                        0L,                            MCI_END_CONSTANT,
"track \0"                   MCI_TRACK,                     MCI_INTEGER,
"\0",                        0L,                            MCI_END_COMMAND,
```

图 2-18　MCI_STATUS 命令清单

字符串句法分析程序生成的 pParam2 **参数如下图所示**:

```
{
HWND       hwndCallback;
ULONG      ulReturn;
ULONG      ulItem;
ULONG      ulTrack;
}
```

注意:MCI_CONSTANT **块填入了** status **结构的** ulItem **字段。如果** MCI_CONSTANT **对应行的关键词字段是**" \0"**或** NULL,**那么句法分析程序将查找块中定义的任一个关键词,第一个匹配的** ULONG **型关键词将被放入** ulItem **字段。如果** MCI_CONSTANT **对应行的关键词不是**" \0",**那么句法分析程序将查找该关键词,并将该行的第一个** ULONG **与** ulParam1 **作或运算。然后,句法分析程序继续在常数块中查找匹配的关键词。例如**:

```
"capability",        MCI_GETDEVCAPS,              0,    MCI_COMMAND_HEAD,
"",                  MCI_PREROLL_TYPE_RETURN,     0,    MCI_RETURN_TYPE,
  "deterministic",   MCI_PREROLL_DETERMINISTIC,         MCI_RETURN_TYPE_STRING,
  "notified"         MCI_PREROLL_NOTIFIED,              MCI_RETURN_TYPE_STRING,
  "none",            MCI_PREROLL_NONE,                  MCI_RETURN_TYPE_STRING,
"none",              0L,                                MCI_END_RETURN_TYPE,
"",                  MCI_TRUE_FALSE_RETURN,       0,    MCI_RETURN_TYPE,
 "TRUE",             1L                                 MCI_RETURN_TYPE_STRING,
 "FALSE",            0L,                                MCI_RETURN_TYPE_STRING,
"",                  0L,                                MCI_END_RETURN_TYPE,
"",                  MCI_DEVICENAME_RETURN,       0,    MCI_RETURN_TYPE,
 "videotape",        MCI_DEVTYPE_VIDEOTAPE,       0,    MCI_RETURN_TYPE_STRING,
 "videodisc",        MCI_DEVTYPE_VIDEODISC,       0,    MCI_RETURN_TYPE_STRING,
 "CDaudio",          MCI_DEVTYPE_CD_AUDIO,        0,    MCI_RETURN_TYPE_STRING,
```

```
    "DAT",                  MCI_DEVTYPE_DAT,              0,    MCI_RETURN_TYPE_STRING,
    "Audiotape",            MCI_DEVTYPE_AUDIO_TAPE,       0,    MCI_RETURN_TYPE_STRING,
    "Other",                MCI_DEVTYPE_OTHER,            0,    MCI_RETURN_TYPE_STRING,
    "Waveaudio",            MCI_DEVTYPE_WAVEFORM_AUDIO,   0,    MCI_RETURN_TYPE_STRING,
    "Sequencer",            MCI_DEVTYPE_SEQUENCER,        0,    MCI_RETURN_TYPE_STRING,
    "Ampmix",               MCI_DEVTYPE_AUDIO_AMPMIX,     0,    MCI_RETURN_TYPE STRING,
    "Overlay",              MCI_DEVTYPE_OVERLAY,          0,    MCI_RETURN_TYPE_STRING,
    "Digitalvideo",         MCI_DEVTYPE_DIGITAL_VIDEO,    0,    MCI_RETURN_TYPE_STRING,
    "Speaker",              MCI_DEVTYPE_SPEAKER,          0,    MCI_RETURN_TYPE_STRING,
    "Headphone",            MCI_DEVTYPE_HEADPHONE,        0,    MCI_RETURN_TYPE_STRING,
    "Microphone",           MCI_DEVTYPE_MICROPHONE,       0,    MCI_RETURN_TYPE_STRING,
    "Monitor",              MCI_DEVTYPE_MONITOR,          0,    MCI_RETURN_TYPE_STRING,
    "CDXA",                 MCI_DEVTYPE_CDXA,             0,    MCI_RETURN_TYPE_STRING,
  "",                       0L,                                 MCI_END_RETURN_TYPE,
  "",                       MCI_INTEGER,                        MCI_RETURN,
  "notify",                 MCI_NOTIFY,                         MCI_FLAG,
  "wait",                   MCI_WAIT,                           MCI_FLAG,
  "",                       MCI_GETDEVCAPS_ITEM,                MCI_CONSTANT,
    "can record",           MCI_GETDEVCAPS_CAN_RECORD,          MCI_INTEGER,
    "can insert",           MCI_GETDEVCAPS_CAN_RECORD_INSERT,   MCI_INTEGER,
    "has audio",            MCI_GETDEVCAPS_HAS_AUDIO,           MCI_INTEGER,
    "has video",            MCI_GETDEVCAPS_HAS_VIDEO,           MCI_INTEGER,
    "can eject",            MCI_GETDEVCAPS_CAN_EJECT,           MCI_INTEGER,
    "can play",             MCI_GETDEVCAPS_CAN_PLAY,            MCI_INTEGER,
    "can save",             MCI_GETDEVCAPS_CAN_SAVE,            MCI_INTEGER,
    "uses files",           MCI_GETDEVCAPS_USES_FILES,          MCI_INTEGER,
    "compound device",      MCI_GETDEVCAPS_USES_FILES,          MCI_INTEGER,
    "can lockeject",        MCI_GETDEVCAPS_CAN_LOCKEJECT,       MCI_INTEGER,
    "can setvolume",        MCI_GETDEVCAPS_CAN_SETVOLUME,       MCI_INTEGER,
    "preroll type",         MCI_GETDEVCAPS_PREROLL_TYPE,        MCI_INTEGER,
    "preroll time",         MCI_GETDEVCAPS_PREROLL_TIME,        MCI_INTEGER,
    "device type",          MCI_GETDEVCAPS_DEVICE_TYPE,         MCI_INTEGER,
    "can stream",           MCI_GETDEVCAPS_CAN_STREAM,          MCI_INTEGER,
    "can process internal", MCI_GETDEVCAPS_CAN_PROCESS_INTERNAL, MCI_INTEGER,
  "",                       0L,                                 MCI_END_CONSTANT,
  "message",                MCI_GETDEVCAPS_MESSAGE,             MCI_CONSTANT,
    "acquire",              MCI_ACQUIREDEVICE,            0,    MCI_INTEGER,
    "release",              MCI_RELEASEDEVICE,            0,    MCI_INTEGER,
    "open",                 MCI_OPEN,                     0,    MCI_INTEGER,
    "close",                MCI_CLOSE,                    0,    MCI_INTEGER,
    "escape",               MCI_ESCAPE,                   0,    MCI_INTEGER,
    "play",                 MCI_PLAY,                     0,    MCI_INTEGER,
```

```
    "seek",              MCI_SEEK,                    0,        MCI_INTEGER,
    "stop",              MCI_STOP,                    0,        MCI_INTEGER,
    "pause",             MCI_PAUSE,                   0,        MCI_INTEGER,
    "info",              MCI_INFO,                    0,        MCI_INTEGER,
    "capability",        MCI_GETDEVCAPS,              0,        MCI_INTEGER,
    "status",            MCI_STATUS,                  0,        MCI_INTEGER,
    "spin",              MCI_SPIN,                    0,        MCI_INTEGER,
    "set",               MCI_SET,                     0,        MCI_INTEGER,
    "step",              MCI_STEP,                    0,        MCI_INTEGER,
    "record",            MCI_RECORD,                  0,        MCI_INTEGER,
    "sysinfo",           MCI_SYSINFO,                 0,        MCI_INTEGER,
    "save",              MCI_SAVE,                    0,        MCI_INTEGER,
    "cue",               MCI_CUE,                     0         MCI_INTEGER,
    "update",            MCI_UPDATE,                  0,        MCI_INTEGER,
    "setcuepoint",       MCI_SET_CUEPOINT,            0,        MCI_INTEGER,
    "setpositionadvise", MCI_SET_POSITION_ADVISE,     0,        MCI_INTEGER,
    "setsyncoffset",     MCI_SET_SYNC_OFFSET,         0,        MCI_INTEGER,
    "load",              MCI_LOAD,                    0         MCI_INTEGER,
    "masteraudio",       MCI_MASTERAUDIO,             0         MCI_INTEGER,
    "gettoc",            MCI_GETTOC,                  0,        MCI_INTEGER,
    "connector",         MCI_CONNECTOR,               0,        MCI_INTEGER,
    "resume",            MCI_RESUME,                  0,        MCI_INTEGER,
   "",                   0L,                                    MCI_END_CONSTANT,
 "",                     0L,                                    MCI_END_COMMAND,
```

pParam2 参数会是这个样子:

```
{
ULONG       hwndCallback;
ULONG       ulReturn;
ULONG       ulItem;
ULONG       ulMessage;
}
```

注意:那个关键词为“message”的第二个常数块。句法分析程序将查找一个形如“message open”的字符串。这个常数把 MCI_GETDEVCAPS_MESSAGE 与 ulParam1 相或,并将值 MCI_OPEN 放入 ulMessage 字段。

多媒体字符串句法分析程序还支持一个默认的 INTEGER 和 STRING 值。它使用常数 MCI_DEFAULT_INTEGER 和 MCI_DEFAULT 来定位不明类型的关键词。例如,如果一条命令需要一个文件名作为其唯一的参数,那么命令表中将会用到下面的一行:

```
"\0",MCI_FILENAME,     MCI_DEFAULT_STRING,
```

这在结构中占用了一个 ULONG,与 MCI_STRING 的情形一样。

对设备特有命令表的支持,是由多媒体安装程序提供的。当安装一个设备的时候,命令表(资源 DLL)是安装过程中所需的参数之一。而必要命令表和设备类型命令表则由系统提供。

2.5.3 错误表

除了用户自定义的命令表之外,MCD 还可以拥有用户自定义的错误表。错误表对驱动程序来说是资源。它们是由 RCDATA 资源类型创建的。在默认情况下,用户自定义的驱动程序错误表是与 MMPM2.INI 文件中说明的 MCDTABLE 和 VSDTABLE 两个 DLL 文件相对应的。资源号等于设备类型号 + MMERROR_TABLE_BASE。例如,下面的例子中一个 RCDATA 值为 506,它由两部分组成:

· 500,来自位于 MCIDRV.H 文件中的 MMERROR_TABLE_BASE。

· 6,来自你所支持的设备类型。(例如,位于 MCIOS2.H 文件中的 MCI_DEVTYPE_OTHER。)

另一种办法是,你可以在你的头文件中增加一条 # define 说明语句,来定义 RCDATA 值。例如:

```
#define MMERR_TABLE_MY_DEVICE     MMERROR_TABLE_BASE + MCI_DEVTYPE_OTHER
```

一旦用该驱动程序产生的错误调用 mciGetErrorString,MDM 就要使用这些表。如果用户自定义的错误表不存在,那么 MDM 将使用默认的错误表。错误表是一个资源。

错误表由若干 ULONG 字符串组成,并有一个特殊的表尾标识符,如下面的例子所示:

```
RCDATA          506
BEGIN

MCIERR_INVALID_DEVICE_NAME,        "Invalid Device Name given"
MCIERR_SUCCESS,                    "MMPM Command completed successfully"
MCIERR_INVALID_DEVICE_ID,          "Invalid device ID given"
MCIERR_UNRECOGNIZED_KEYWORD,       "Unrecognized keyword"
MCIERR_UNRECOGNIZED_COMMAND,       "Unrecognized command"
MCIERR_HARDWARE,                   "Hardware error"
```

```
MCIERR_OUT_OF_MEMORY,              "System out of memory"
      .
      .
      .
MCIERR_MSG_TABLE_END               ""
END
```

注意：如果设备不能使用某个基本命令，MCD 会返回 MCIERR_UNSUPPORTED_FUNCTION。如果设备支持该命令，但不是所有的选项都支持，MCD 可能会返回 MCIERR_UNSUPPORTED_FLAG。

2.6 设备状态

任何传输数据的媒体设备在一时刻，总是处于以下的九种状态之一：

· 播放
· 录制
· 查找
· 停止
· 播放过程中的暂停
· 录制过程中的暂停
· 为播放操作做提示
· 为录制操作做提示
· 关闭

当一个设备刚刚打开的时候，它的设备上下文总是处于停止状态的。所谓的关闭状态可以看作既指初始状态又指终止状态，或者可以干脆认为根本就不存在所谓的关闭状态，因为设备上下文在打开之前并不存在，而且在关闭之后亦不复存在。

研究一下图 2-19 的设备状态表。表的第一列列出的是所有可能的设备状态。这个表给出了当收到顶上一行中的命令消息后设备状态发生的相应变化。表中假定任何错误的操作都不改变设备的当前状态。例如，若一个波形播放器是打开的，但此刻不存在任何元素供它播放，如果向这个波形播放器发出播放命令，它将继续保持在停止状态，而 MCD 会收到错误码。

注意：仅当进行查找(SEEK)操作时，设备才会处在查找状态。当完成查找后，设备立即进入停止状态。应用程序开发者和 MCD 的编写者可以把图 2-19 作为参考。尽管不能保证每一种媒体设备的操作都与这个表相符，但应用程序应当尽量使设备的操作看起来简单。

状态 \ 消息		CLOSE	CUE|PLY	CUE|REC	LOAD	OPEN	PAUSE	PLAY	RECORD	RESUME	SEEK	SP|N|UP	SP|N|DN	STEP	STOP
1	PLAY	9	7	8	4	-	5	1	2	1	3	1	4	5	4
2	RECORD	9	7	8	4	-	6	1	2	2	3	-	4	6	4
3	SEEK	9	7	8	4	-	3	1	2	3	3	7	4	E	4
4	STOP	9	7	8	4	-	4	1	2	4	3	7	4	4	4
5	PAUSE_PLAY	9	7	8	4	-	5	1	2	1	3	7	4	5	4
6	PAUSE_REC	9	7	8	4	-	6	1	2	2	3	-	4	5	4
7	CUE_PLAY	9	7	8	4	-	7	1	2	7	3	7	4	7	4
8	CUE_REC	9	7	8	4	-	8	1	2	8	3	-	4	8	4
9	CLOSE	-	-	-	-	4	-	-	-	-	-	-	-	-	-

图 2-19 设备状态表

2.7 控制流式设备:波形音频 MCD

二重唱演奏器 I 样例程序调用 mciSendCommand 函数,向媒体设备管理器(MDM)发送命令消息,以播放用户硬盘上的波形文件。应用程序不必知道硬件的细节,也就是说它无需知道用来播放波形文件的究竟是哪一种声卡。MDM 只是将命令发往一个通用的音频设备接口——mciDriverEntry。这个接口位于 AUDIOMCD DLL 中。这个设备无关的模块 AUDIOMCD 加上另一个设备相关的模块 AUDIOIF,组成波形音频 MCD,如图2-20所示。其中还可以看到 MCD 所用到的 MMIO 和 SSM 两个接口。

图 2-20 波形音频 MCD 模块

2.7.1 波形音频到混响放大器的连接

为了实现播放和录制波形数据这一基本目标,波形音频 MCD 要发送 MCI_SET 应用程序请求以设定音频属性。波形音频 MCD 之所以能够向混响放大器发送请求,是因为从波形音频 MCD 到混响放大器 MCD 之间存在一个默认的逻辑连接。

所连接的设备的名称就是一个默认的连接,而一个设备上下文连接是一个打开状态的设备的一个特定实例的实际句柄。波形音频设备有一个到混响放大设备的默认连接。当一个波形音频设备打开时,它会自动打开混响放大设备,为两个设备各创建一个实例。由于在 OS/2 多媒体系统中可以共享设备,所以波形音频设备可以被另一个应用程序再次打开,而这样就会又有两个新的实例被创建出来。这时,对这前后两者来说,默认连接是相同的。而设备上下文连接则是不同的两个。

图 2-21 中的代码节选描述了波形音频设备怎样获取默认连接并打开一个相应的混响放大器。波形音频 MCD 使用混响放大器的设备句柄来发送申请以改变音量。

应用程序可以通过发送 MCI_CONNECTION 消息来获取混响放大器的句柄。如果应用程序需要使用任何高级的音频整形功能,比如高音(treble),低音(bass)和平衡(balance)等等,那么它必须获取这个句柄。

```
ulpInstance->usWaveDeviceID = pDrvOpenParams->usDevlceID;
ulDeviceTypeID = MAKEULONG(MCI_DEVTYPE_WAVEFORM_AUDIO,
                           pDrvOpenParams->usDeviceOrd);
/****************************
 * Ensure that the INI file contains the right device id
 ****************************/
if (pDrvOpenParams->usDeviceType != MCI_DEVTYPE_WAVEFORM_AUDIO)
   {
   return (MCIERR_INI_FILE);
   }
usConnLength = sizeof(DEFAULTCONNECTIONS2);
ulrc = mciQueryDefaultConnections(ulDeviceTypeID,
                                  &DefCon,
                                  &usConnLength);
/****************************
 * Ensure that the INI file says that we are connected
 * to an amp mixer. If it says that we are connected
 * to ourselves, return an error.
 ****************************/
if (ULONG_LOWD(DefCon.dwDeviceTypeID2) == MCI_DEVTYPE_WAVEFORM_AUDIO)
   {
   return(MCIERR_INI_FILE);
   }
```

图 2-21 ConnectToAmp 子例程(LOADSUBS.C)

2.7.2 MMIO 操作

波形音频 MCD 使用 MMIO 函数来完成下列功能:

1. 打开波形数据文件;
2. 用从该文件的文件头中得到的信息初始化音频设备;
3. 支持多种文件格式;
4. 支持网络功能;
5. 支持编辑功能。

2.7.2.1 初始化音频设备

为了初始化音频设备,波形音频 MCD 需要知道关于要播放的数据元素的信息。驱动程序调用 mmioGetHeader,后者返回波形元素文件头中的信息(如录制的数据是立体声的还是单声道的,它的采样频率及每次采样的位数是多少,等等)。AUDIOMCD 分析这些信息,并把它放入实例的结构中。然后,这个实例结构被送往 AUDIOIF。AUDIOIF 通过 IOCtl 接口,使用这些信息来初始化设备。

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * *
 * A streaming MCD should utilize MMIO to perform all
 * file manipulations . If we use MMIO,then the MCD
 * will be free from file dependencies,that is,if a RIFF
 * IOProc or a VOC IOProc is loaded will be irrelevant .
 * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulrc = mmioGetHeader(ulpInstance - > hmmio,
                     (PVOID)&ulpInstance - > mmAudioHeader,
                     sizeof(ulpInstance - > mmAudioHeader),
                     (PLONG)&BytesRead,
                     (ULONG) NULL,
                     (ULONG) NULL);
if (ulrc = = MMIO_SUCCESS)
    {
    / * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Copy the data from the call into the instance
     * so that we can set the amp/ mixer up with the
     * values that the file specifies .
     * * * * * * * * * * * * * * * * * * * * * * * * * */

     AMPMIX .sMode               = WAVEHDR .usFormatTag;
     AMPMIX .sChannels           = WAVEHDR .usChannels;
     AMPMIX .lSRate              = WAVEHDR .ulSamplesPerSec;
     AMPMIX .lBitsPerSRate       = WAVEHDR .usBitsPerSample;
     ulpInstance - > ulDataSize = XWAVHDR .ulAudioLengthInBytes;
     AMPMIX .ulBlockAlignment = (ULONG)WAVEHDR .usBlockAlign;
```

```
    ulpInstance - > ulAverageBytesPerSec = WAVEHDR.usChannels *
      WAVEHDR.ulSamplesPerSec * (WAVEHDR.usBitsPerSample/ 8);

    }/ * SuccesFul GetHeader */
else
  {
  ulrc = mmioGetLastError(ulpInstance - > hmmio);

  }
  return(ulrc);

}/ * GetAudioHeader */
```

图 2-22 GetAudioHeader 子例程(AUDIOSUB.C)

传送给 mmioGetHeader 的句柄 hmmio 由 mmioOpen 返回,以打开数据元素,当控制由 mmioGetHeader 返回到 MMIO 管理器时,MMIO 管理器将填充由 mmAudioHeader 所指向的缓冲区。这是一个 MMAUDIODATA 结构。当 mmioGetHeader 返回这个 MMAUDIODATA 结构后,这个结构中已包含了波形音频 MCD 初始化音频设备所需要的全部信息。这些信息包括:

1 . 文件中的波形数据的类型(usFormatTag);

2 . 文件是单声道的还是立体声的(usChannels);

3 . 文件录制时的采样频率及每次采样的位数(usSamplesPerSec 和 usBitsPerSample);

4 . 音频数据的长度(ulAudioLengthInBytes)。

2.7.2.2 支持多种文件格式

图 2-23 显示了波形音频 MCD 如果通过在 MMCTOCENTRY 数据结构的 ulMedType 字段定义 MMIO_MEDIATYPE_AUDIO 来搜索音频 I/ O 过程。为了搜索和标识其它类型的 I/ O 过程,你可以包括下列媒体类型标志之一:

1 . MMIO_MEDIATYPE_IMAGE;

2 . MMIO_MEDIATYPE_AUDIO;

3 . MMIO_MEDIATYPE_MIDI;

4 . MMIO_MEDIATYPE_COMPOUND;

5 . MMIO_MEDIATYPE_OTHER;

6 . MMIO_MEDIATYPE_UNKNOWN;

7 . MMIO_MEDIATYPE_DIGITALVIDEO;

8 . MMIO_MEDIATYPE_ANIMATION;

9 . MMIO_MEDIATYPE_MOVIE。

欲了解进一步的信息,可参阅《OS/ 2 多媒体编程参考》。

```
    mmioinfo.aulInfo[3] = MMIO_MEDIATYPE_AUDIO;

   / * Open the file */
```

```
    pInstance->hmmio = mmioOpen(pFileName,
                                &mmioinfo,
                                ulFlags);

    /* Check for errors--see comments from above */

    if (pInstance->hmmio == (ULONG)NULL)
       {
       if (mmioinfo.ulErrorRet == MMIOERR_MEDIA_NOT_FOUND)
          {
          return(MCIERR_INVALID_MEDIA_TYPE);
          }

       return(mmioinfo.ulErrorRet);
       }

    pInstance->ulCapabilities = 0;

 else
    {
    /**************************
     * Since the wave IOProc opened the file, we know
     * that it has the following capabilities.
     **************************/
    pInstance->ulCapabilities = (CAN_INSERT|CAN_DELETE|CAN_UNDOREDO +
                                 CAN_SAVE|CAN_INSERT|CAN_RECORD);
    }

/**************************
 * Get The Header Information
 **************************/

 if (!(ulFlags & MMIO_CREATE))
    {
    ulrc = GetAudioHeader(pInstance);

    } /* Not Create Flag */
 else
    {
    pInstance->ulDataSize = 0;
    }

 pInstance->fFileExists = TRUE;

/**************************
 * You cannot do the set header immediately after file creation
 * because future sets on samples, bitpersample,channels may follow
 **************************/

 return(ulrc);
 } /* OpenFile */
```

图 2-23　OpenFile 子例程(AUDIOSUB.C)

2.7.2.3 网络功能

一些 OS/ 2 网络具有支持网络功能的 I/ O 过程。MMPM2 .INI 文件中包含一些信息,描述了当通过网络来播放或录制文件时数据流的质量。图 2-24 显示了如何从这个 INI 文件中获取服务质量(Quality of Service,简写为 QOS)的值。

如果你在编写一个流 MCD,使用 MMIOM_BEGINSTREAM 和 MMIOM_ENDSTREAM 消息可以改善在局域网(LAN)上的执行状况。在图 2-24 的例子中,使用了 MMIOM_BEGINSTREAM 消息来告诉 I/ O 过程我要通过网络来传递数据流。

```
/* -------------------------------------------------------
 * The MMPM2.INI file contains two variables that
 * a streaming MCD should retrieve.The first one
 * QOS_VALUE(Quality of Service)contains settings
 * which describe the quality of service that the
 * network the user is streaming from will try to
 * support (for example, GUARANTEED or DONTCARE) .
 * If this quality of service is not available, then another
 * variable (QOSERRORFLAG)describes whether or not
 * to notify the caller .
 ------------------------------------------------------- */

ulrc = mciQuerySysValue(MSV_SYSQOSVALUE, &ulpInstance -> lQosValue);
if ( ! ulrc)
   {
   ulpInstance -> lQosValue = DONTRESERVE;
   }
ulrc = mciQuerySysValue(MSV_SYSQOSERRORFLAG, &ulpInstance -> lQOSReporting);
if ( ! ulrc)
   {
   ulpInstance -> lQOSReporting = ERROR_DEFAULT;
   }
```

图 2-24 Quality of Service 子例程(LOADSUBS .C)

图 2-25 给出了 EndQuality of Service 子例程函数,一些 OS/ 2 网络具有特殊的支持网络功能的 I/ O 过程。这个例子使用了 MMIOM_BEGINSTREAM 和 MMIOM_ENDSTREAM 消息来告诉这些 I/ O 过程,波形音频 MCD 要通过网络来传递数据流。

```
ULONG EndQualityofService(INSTANCE     * pInstance)
{
```

```
    LONG      rc;

      rc = mmioSendMessage(pInstance - > hmmio,
                           MMIOM_ENDSTREAM,
                           0,
                           0);

      return(rc);
    }/ * EndQualityofService */
```

图 2-25　EndQuality of Service 子例程(STRMSUBS.C)

2.7.2.4　使用 MMIO 编辑功能

本节讨论了 MCD 如何使用 MMIO 的编辑功能来实现媒体控制接口(MCI)的编辑功能。表 2-12 列出了这些 MMIO 功能以及它们在媒体控制接口中的等价功能。

表 2-12　MMIO 功能表

MMIO 功能	MCI 中的等价功能
MMIOM_DELETE	MCI_CUT
MMIOM_READ	MCI_READ
MMIOM_DELETE	MCI_DELETE
MMIOM_UNDO	MCI_UNDO
MMIOM_REDO	MCI_REDO
MMIOM_DELETE,MMIOM_BEGININSERT 及 MMIOM_ENDINSERT	MCI_PASTE

图 2-26 显示了怎样利用 MMIO 功能将数据粘贴(paste)到一个文件中。

```
   / * * * * * * * * * * * * * * * * * * * * * * * * * *
    * Remove the information in the file if
    * from/ to are specified .
    * * * * * * * * * * * * * * * * * * * * * * * * * */

    if (ulParaml&MCI_FROM| |ulParaml&MCI_TO)
      {
      lReturnCode = mmioSendMessage(plnstance - > hmmio,
                                    MMIOM_DELETE,
                                    pEditParms - > ulFrom,
                                    ulPasteLen);

       if (lReturnCode ! = MMIO_SUCCESS)
          {
          ulrc = mmioGetLastError(pInstance - > hmmio);
```

```
        PasteNotify( &FuncBlock,ulParaml,ulrc);
        return(ulrc);
        }
    }
  if ( !(ulParaml & MCI_TO_BUFFER))
    {
    /***************************
     * Let the IOProc know that the information is
     * about to be inserted into the file
     ***************************/

    lReturnCode = mmioSendMessage(pInstance->hmmio,
                                  MMIOM_BEGININSERT,
                                  0,
                                  0);
   if (lReturnCode != MMIO_SUCCESS)
      {
      ulrc = mmioGetLastError(pInstance->hmmio);
      PasteNotify( &FuncBlock,ulParaml,ulrc);
      return(ulrc);
      }
   /***************************
    * Write the information that we received from
    * the clipboard
    ***************************/

   lReturnCode = mmioWrite(pInstance->hmmio,
                           (PSZ)pBuffer,
                           ulBuffLen);
   if (lReturnCode == MMIO_ERROR)
      {
      ulrc = mmioGetLastError(pInstance->hmmio);
      PasteNotify( &FuncBlock,ulParaml,ulrc);
      return(ulrc);
      }
   /***************************
    * We have finished inserting the information
    * the paste is complete
    ***************************/

   lReturnCode = mmioSendMessage(pInstance->hmmio,
                                 MMIOM_ENDINSERT,
                                 0,
                                 0);
```

```
if ( lReturnCode ! = MMIO_SUCCESS)
   {
   ulrc = mmioGetLastError(pInstance - > hmmio);
   PasteNotify( &FuncBlock,ulParaml,ulrc);
   return(ulrc);
   }
```

图 2-26　StartPaste 子例程(ADMCPST .C)

2 .7 .3　同步/ 流操作

波形音频 MCD 利用 SPI 函数与同步/ 流管理器(SSM)通信,为传递数据流做准备,然后把数据从源位置(缓冲区)传到目标位置(声卡)。

2 .7 .3 .1　准备传递波形数据流

当波形音频 MCD 收到了一个打开设备的请求时,它调用 SpiGetHandler 函数,把源流处理器和目标流处理器的名称传给 SSM,以使它知道哪两个流处理器要传递数据。只有当 SSM 知道了这个信息,数据流动才能开始。

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Get 'A 'stream Handler Handles for Source & target operations
 * The file system stream handler is the default 'A ' handler
 * but the memory stream handler will be used for playlists .
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

   if (ulrc = SpiGetHandler((PSZ)DEFAULT_SOURCE_HANDLER_NAME,
                             &hidASource,
                             &hidATarget))
     {
     return(ulrc);
     }

   / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    * Get 'B ' stream Handler Handles for Source & target operations
    * The audio stream handler is considered the B stream handler
    * since it will usually be the target .
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

   ulrc = SpiGetHandler((PSZ)DEFAULT_TARGET_HANDLER_NAME,
                         &hidBSource,
                         &hidBTarget);
   return(ulrc);
```

图 2-27　StreamSetup 子例程(LOADSUBS .C)

当波形音频 MCD 获得了流处理器的 ID 号后,它必须填充 SPCBKEY 数据结构,以告诉 SSM 将要传递的是何种数据,SPCBKEY 结构有 3 个字段:一个数据类型和两个子数据类型。例如,它可能会告诉 SSM:“我要传递 PCM 数据,8 位,22kHz”。

然后,波形音频 MCD 必须填充设备控制块(DCB),以告诉 SSM 数据要送往哪个设备(如图 2-28 所示)。DCB 包含两项基本信息:

1 . ASCII 字符串形式的设备名称;

2 . 设备的句柄。

ASCII 字符串形式的设备名称是从 INI 文件中获得的,并且由 MDM 传递过来。当 AUDIOIF 打开设备后,IOCtl 返回一个系统文件号,这就是设备的句柄。

```
SysInfo.ulItem          = MCI_SYSINFO_QUERY_NAMES;
SysInfo.usDeviceType = LOUSHORT(ulDeviceType);
SysInfo.pSysInfoParm = &QueryNameParm;

itoa(HIUSHORT(ulDeviceType),szIndex,10);

szIndex[1] = '\0';

strncat(szAmpMix,szIndex,2);
strcpy(QueryNameParm.szLogicalName,szAmpMix);

if(rc = mciSendCommand(0,
                    MCI_SYSINFO,
                    MCI_SYSINFO_ITEM|MCI_WAIT,
                    (PVOID)&SysInfo,
                    0))
      return(rc);
/**************************
 * Get PDD associated with our AmpMixer
 * Device name is in pSysInfoParm->szPDDName
 **************************/

SysInfo.ulItem       = MCI_SYSINFO_QUERY_DRIVER;
SysInfo.usDeviceType   = (USHORT)ulDeviceType;
SysInfo.pSysInfoParm = &SysInfoParm;

strcpy(SysInfoParm.szInstallName,QueryNameParm.szInstallName);

if(rc = mciSendCommand (0,
                    MCI_SYSINFO,
                    MCI_SYSINFO_ITEM|MCI_WAIT,
                    (PVOID)&SysInfo,
                    0))
    return(rc);
```

```
  strcpy(szPDDName,SysInfoParm.szPDDName);

  return(MCIERR_SUCCESS);

} /* GetPDDName */
```

图 2-28 GetPDDName 子例程(AUDIOSUB.C)

2.7.3.2 创建数据流

在获得了流处理器的 ID 号。并填充了 SPCBKEY 和 DCB 信息之后, MCD 调用 SpiCreateStream 函数,以完成这样几件工作:初始化两个流处理器以创建数据流,为缓冲区分配内存,并设置进入传送数据流状态。

图 2-29 中的例子示范了如何创建数据流。调用者要提供源流处理器和目标流处理器,并且预先填充好音频设备控制块(见 MCIOPEN.C)。调用者还要提供事件过程,以通知所有的流事件。

```
ulrc = SpiCreateStream(hidSrc,
                       hidTgt,
                       &pInstance->StreamInfo.SpcbKey,
                       (PDCB)&pInstance->StreamInfo.AudioDCB,
                       (PDCB)&pInstance->StreamInfo.AudioDCB,
                       (PIMPL_EVCB)&pInstance->StreamInfo.Evcb,
                       (PEVFN)EventProc,
                       (ULONG)NULL,
                       hStream,
                       &pInstance->StreamInfo.hEvent);
```

图 2-29 CreateNAssocStream 子例程(AUDIOSUB.C)

当 MCD 创建一个数据流的时候,它必须向 SSM 注册一个回调句柄。这个回调句柄是 MCD 代码中的一个入口点或函数,它用来处理在数据流动过程中从 SSM 返回来的事件。这个回调句柄是一个强有力的机制,因为它解放了驱动程序,使驱动程序得以在数据流动过程中从事其它工作。当一个事件发生时,由 SSM 检测到它,并把它通知到回调地址。

这个波形音频 MCD 范例包括下列事件例程(routine):

- RecordEventRoutine(ADMCRECD.C),如图 2-30;
- PlayEventRoutine(ADMCPLAY.C),如图 2-31。

```
RC APIENTRY RecordEventRoutine(MEVCB   * pevcb)
{
  MTIME_EVCB        * pMTimeEVCB;     // Modified EVCB
  INSTANCE          * ulpInstance;    // Instance Ptr
```

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * *
 * EventProc receives asynchronous SSM event notifications
 * When the event is received,the event semaphore is posted
 * which will wake up the MCD thread(s) blocked on this
 * semaphore .
 * The semaphore is not posted for time events like
 * cuepoint(TIME)and media position changes since they do
 * not alter the state of the stream .
 * * * * * * * * * * * * * * * * * * * * * * * * * */

switch(pevcb - > evcb.ulType)
{
case EVENT_IMPLICIT_TYPE:

 / * Retrieve our instance from the EVCB */

  ulpInstance = (INSTANCE * )pevcb - > ulpInstance;

  switch(pevcb - > evcb.ulSubType)

  {
  case EVENT_ERROR:
       ulpInstance - > StreamEvent = EVENT_ERROR;


       / * * * * * * * * * * * * * * * * * * * * * * * * * *
        * Check for playlist specific error first
        * * * * * * * * * * * * * * * * * * * * * * * * * */

       / * * * * * * * * * * * * * * * * * * * * * * * * * *
        * End of PlayList event is received
        * as an implicit error event .It
        * is treated as a normal EOS
        * * * * * * * * * * * * * * * * * * * * * * * * * */
       if(ulpInstance - > usPlayLstStrm = = TRUE)
          if (pevcb - > evcb.ulStatus = = ERROR_END_OF_PLAYLIST)
              ulpInstance - > StreamInfo.Evcb.evcb.ulStatus =
                   MMIOERR_CANNOTWRITE;
       DosPostEventSem(ulpInstance - > hEventSem);
       break;

  case EVENT_STREAM_STOPPED:
       / * * * * * * * * * * * * * * * * * * * * * * * * * *
        * Event Stream Stopped.Release the
        * Blocked thread
```

```
        * * * * * * * * * * * * * * * * * * * * * * * * * */
        ulpInstance - > StreamEvent = EVENT_STREAM_STOPPED;
        DosPostEventSem(ulpInstance - > hEventSem);
      break;
    case EVENT_SYNC_PREROLLED:
        / * * * * * * * * * * * * * * * * * * * * * * * * * *
         * This event is received in response to a
         * preroll start .A Preroll start is done
         * on an MCI_CUE message .
         * * * * * * * * * * * * * * * * * * * * * * * * * */
        ulpInstance - > StreamEvent = EVENT_SYNC_PREROLLED;
        DosPostEventSem(ulpInstance - > hEventSem);
      break;
  .
  .
  .
```

图 2-30 RecordEventRoutine 子例程(ADMCRECD .C)

Record Event Routine 中的 MEVCB 结构是一个经过修改的 IMPL_EVCB,SSM 使用它来通知事件(参阅《OS/ 2 多媒体编程参考》)。你可以增加额外的字段来扩展 EVCB 结构。例如,图 2-31 中的 MEVCB 增加了一个实例指针,其中包含了本地实例数据。当 SSM 调用波形音频 MCD 的时候,它允许 MCD 存取本地实例数据。

图 2-31 所示的 PlayEventRoutine 假定可以接收来自 SSM 的所有类型的事件通知,其中包括隐式事件和提示点(cue point)(以时间和数据形式给出)。作为对提示点通知的响应,由函数 mdmDriverNotify 向 MDM 返回一条 MCI_CUEPOINT 消息。

```
RC APIENTRY PlayEventRoutine(MEVCB     * pevcb)
{
 MTIME_EVCB        * pMTimeEVCB;       / * Modified Time EVCB */
 INSTANCE          * ulpInstance;      / * Current Instance */
 HWND              hWnd;               / * Callback Handle */
 BOOL              fPlayListDone = FALSE;

  / * * * * * * * * * * * * * * * * * * * * * * * * * *
   * EventProc receives asynchronous SSM event notifications
   * When the event is received, the event semaphore is posted
   * which will wake up the MCD thread(s) blocked on this
   * semaphore .
   * The semaphore is not posted for time events like
   * cuepoint(TIME)and media position changes since they do
   * not alter the state of the stream .
```

```
 **************************/

  switch(pevcb->evcb.ulType)
  {
  case EVENT_IMPLICIT_TYPE:

  /* Retrieve our instance from the EVCB */

  ulpInstance = (INSTANCE *)pevcb->ulpInstance;

  /* Retrieve the callback handle to post messages on */

  hWnd = ulpInstance->hwndCallBack;

  switch(pevcb->evcb.ulSubType)

  {
  case EVENT_EOS:
       ulpInstance->StreamEvent = EVENT_EOS;
       DosPostEventSem(ulpInstance->hEventSem);
     break;

  case EVENT_STREAM_STOPPED:
       /* Self explanatory--someone stopped the stream */


       ulpInstance->StreamEvent = EVENT_STREAM_STOPPED;
       DosPostEventSem(ulpInstance->hEventSem);
     break;

  case EVENT_SYNC_PREROLLED:
       /**************************
        * This event is received in response to a
        * preroll start.A Preroll start is done
        * on an MCI_CUE message.
        **************************/

       ulpInstance->StreamEvent = EVENT_SYNC_PREROLLED;
       DosPostEventSem(ulpInstance->hEventSem);
    break;

case EVENT_PLAYLISTMESSAGE:
       /**************************
        * We can receive this event if a playlist
        * parser hits the MESSAGE COMMAND.
        * NOTE:The MCD should return this message
        * with the callback handle specified on the
```

```
    * open.This could be the source of much
    * grief if you return on the wrong handle.
    ***************************/

    mdmDriverNotify(ulpInstance->usWaveDeviceID,
                    ulpInstance->hwndOpenCallBack,
                    MM_MCIPLAYLISTMESSAGE,
                    (USHORT)MAKEULONG(pevcb->evcb.ulStatus,
                        ulpInstance->usWaveDeviceID),
                    (ULONG) pevcb->evcb.unusedl);
   break;
  case EVENT_PLAYLISTCUEPOINT:

    /***************************
    * We can receive this event if a playlist
    * parser hits the CUEPOINT COMMAND opcode
    * in the playlist.This differs from a "normal"
    * cuepoint because it is detected by the source,
    * rather than the target stream handler.
    ***************************/

    mdmDriverNotify( ulpInstance->usWaveDeviceID,
                     ulpInstance->hwndOpenCallBack,
                     MM_MCICUEPOINT,
                     (USHORT)MAKEULONG(pevcb->evcb.ulStatus,
                         ulpInstance->usWaveDeviceID),
                     (ULONG) pevcb->evcb.unusedl);
    break;
  } /* SubType case of Implicit Events */
  break;

case EVENT_CUE_TIME_PAUSE:
    {
    /***************************
    * This event will arrive if we played to a certain
    * position in the stream.Let the play thread know
    * that we have reached the desired point.
    ***************************/

    pMTimeEVCB = (MTIME_EVCB *)pevcb;
    ulpInstance = (INSTANCE *)pMTimeEVCB->ulpInstance;
    ulpInstance->StreamEvent = EVENT_CUE_TIME_PAUSE;

    DosPostEventSem(ulpInstance->hEventSem);
```

```
        }
        break;
      case EVENT_CUE_TIME:

          break;

  }/ * All Events case */

  return(MCIERR_SUCCESS);

}/ * PlayEventProc */
```

图 2-31　PlayEventRoutine 子例程(ADMCPLAY.C)

SSM 所通知的事件可以是正常事件,比如由于播放完毕产生的数据流结尾;也可以是非常事件,比如在数据流动过程中返回了一个错误信息。驱动程序需要知道这些事件。

SSM 通知两种事件:隐式的及显示式,隐式事件都是要通知的。当发生了一个隐式事件时,SSM 必须向驱动程序通知,但驱动程序并不一定要采取什么行动。

只有当驱动程序请求通知显式事件时,显式事件才会被通知。例如,驱动程序请求通知提示点。

2.7.3.3　事件处理

在数据流创建之前,便可以使用 SpiEnableEvent 函数来允许对隐式和显示事件的通知。图 2-32 描述了 EVENT_CUE_TIME_PAUSE 标志,它的作用是使数据流暂停在提示点上。当在播放或录制过程中数据流到达提示点时,音频流处理器就通知 MCD。注意数据流只是暂停,而不是停止。

```
RC CreateToEvent(INSTANCE * ulpInstance,ULONG ulTo)
{
/ * rename this function CreateToEvent */

  ULONG ulrc;

 / * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Set up a cue time pause event at the place in
  * the stream where the caller wants us to play/ record
  * to.Note:this event will pause the stream and
  * will be considerably more accurate than just
  * setting a cue point,receiving the event and stopping
  * the stream (since a normal cue point will force
  * bleed over).
  * * * * * * * * * * * * * * * * * * * * * * * * * */

  ulpInstance->StreamInfo.TimeEvcb.hwndCallback
       = ulpInstance->hwndCallBack;
```

```
ulpInstance - > StreamInfo .TimeEvcb .usDeviceID
      = ulpInstance - > usWaveDeviceID;
ulpInstance - > StreamInfo .TimeEvcb .evcb .ulType
      = EVENT_CUE_TIME_PAUSE;
ulpInstance - > StreamInfo .TimeEvcb .evcb .ulFlags
      = EVENT_SINGLE;
ulpInstance - > StreamInfo .TimeEvcb .evcb .hstream
      = ulpInstance - > StreamInfo .hStream;
ulpInstance - > StreamInfo .TimeEvcb .ulpInstance
      = (ULONG)ulpInstance;

ulpInstance - > StreamInfo .TimeEvcb .evcb .mmtimeStream = ulTo;


/ * Enable the cue time pause event .*/

ulrc = SpiEnableEvent((PEVCB)&ulpInstance - > StreamInfo .TimeEvcb .evcb,
                      (PHEVENT)&ulpInstance - > StreamInfo .hPlayToEvent);
return(ulrc);

} * CreateToEvent */
```

图 2-32 DoTillEvent 子例程(AUDIOSUB .C)

如果你允许通知某个特定的事件,你必须清除这个事件的句柄,以避免后续的命令使用它。当某个给定的事件被通知后,必须使用函数 SpiDisableEvent 把它显式地清除。一旦一个事件被清除出系统,它就再也不会被检测到或通知给应用程序或 MCD。

```
SpiDisableEvent(ulpInstance - > StreamInfo .hPlayToEvent);
```

图 2-33 StarPlay 子例程(ADMCPLAY .C)

2 .7 .3 .4 关联数据流

在创建一个数据流之后和可以启动它之前,必须确认一下数据资源(或流对象)。图 2-34是一个把数据流关联到一个 MMIO 文件句柄的例子。我们要与数据对象关联的流处理器永远是系统流处理器(FSSH)。这样,如果我们创建了一个播放流,那么 FSSH 就是源,因此与源关联;而对于录制流,FSSH 是目标,所以与目标关联。

```
if(Operation = = PLAY_STREAM)
   {
   ulrc = SpiAssociate( (HSTREAM) * hStream,
                        hidSrc,
                        (PVOID) &pInstance - > StreamInfo .acbmmio);
   } * Associating play stream */
```

图 2-34 CreateNAssocStream 子例程(AUDIOSUB .C)

2.7.3.5 收到 MCI_LOAD 消息后重关联数据流

通常一个数据流需要经过销毁、重建后,才能关联到一个新文件上去。然而,你可以跳过这些步骤而把一个已存在的数据流与一个新文件相关联,前提是新文件要与数据流属于同样的数据类型。以波形音频为例,如果你有一个 16 位 11kB 的 WAVE 流,那么将与这个数据流关联的文件也必须属于相同的数据类型和子类型。

注意:一旦创建了一个数据流,它只能与一个数据对象关联,且后者对应着唯一确定的流处理器(源或目标)。改变关联的时候,数据流不能是活动的——必须首先停止它(突然停止(discard stop)、圆滑停止(flush stop)或遇到流的结尾(EOS))。

```
/****************************
 * Reassociate The Stream Handlers with the new
 * stream object if the stream has been created
 * in the correct direction already.
 ****************************/
if(ulpInstance->ulCreateFlag == PREROLL_STATE)
  {
  /****************************
   * Fill in Associate Control Block Info for
   * file system stream handler(FSSH).FSSH will
   * use the mmio handle we are associating to
   * stream information.
   ****************************/
   ulpInstance->StreamInfo.acbmmio.ulObjType = ACBTYPE_MMIO;
   ulpInstance->StreamInfo.acbmmio.ulACBLen = sizeof(ACB_MMIO);
   ulpInstance->StreamInfo.acbmmio.hmmio = ulpInstance->hmmio;
  /****************************
   * Associate FileSystem as source if Playing.Note
   * the association is always done with the file
   * system stream handler since it is involved with
   * mmio operations.If you try this with the
   * audio stream handler,you will get invalid
   * handle back.
   ****************************/

   if (AMPMIX.ulOperation == OPERATION_PLAY)
     {
     ulrc = SpiAssociate ( ulpInstance->StreamInfo.hStream,
                           ulpInstance->StreamInfo.hidASource,
                           (PVOID)&ulpInstance->StreamInfo.acbmmio);
     }
  /****************************
   * Associate FileSystem as target if recording
```

```
   * * * * * * * * * * * * * * * * * * * * * * * * * */
   else
     {
     ulrc = SpiAssociate ( ulpInstance - > StreamInfo .hStream,
                           ulpInstance - > StreamInfo .hidATarget,
                           (PVOID) & ulpInstance - > StreamInfo .acbmmio);
     }/ * else we are in record mode */
```

图 2-35　MCILOAD .C 子例程(ADMCLOAD .C)

2 .7 .3 .6　对数据流的预处理(prerolling)——出于对运行效果的考虑

预处理一个数据流是指流处理器启动和填充缓冲区。这样可使应用程序启动数据流具有更好的实时响应并触发数据流的同步。这可以由 SpiStartStream 函数通过 SPI_START_PREROLL 标志来做到。

然而,如果数据已处在暂停状态(STOP_PAUSED),表明数据流中已加入了提示(cue),所以就不必调用 SpiStartStream 去重填缓冲区了。图 2-36 显示如何检查数据流是否处于暂停状态。

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * *
 * If the stream is paused, there is no
 * sense in cueing it since the buffers
 * are full anyway
 * * * * * * * * * * * * * * * * * * * * * * * * * */
 if( STRMSTATE = = MCI_PAUSE ||
    STRMSTATE = = STOP_PAUSED)
   {
  / * * * * * * * * * * * * * * * * * * * * * * * * * *
   * * If the stream is going the right way
   * * then we have the ability to avoid the cue
   * * since the buffers have been filled up before
   * * we did the pause
   * * * * * * * * * * * * * * * * * * * * * * * * * */
 if ( AMPMIX .ulOperation = = OPERATION_RECORD & &
      fCueInput)
   {
   ulpInstance - > ulCreateFlag = PREROLL_STATE;
   STRMSTATE = CUERECD_STATE;
   return(MCIERR_SUCCESS);
   }
/ * * * * * * * * * * * * * * * * * * * * * * * * * *
 * If the current stream is cued for playback and
 * we have a cue_output request,our work is done
 * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
    else if (AMPMIX.ulOperation == OPERATION_PLAY &&
             fCueOutput)
       {
       ulpInstance->ulCreateFlag = PREROLL_STATE;
       STRMSTATE = CUEPLAY_STATE;
       return(MCIERR_SUCCESS);
       }
    } /* If the stream may be in cue state */
```

图 2-36 MCICUE.C 子例程(ADMCCUE.C)

2.7.4 波形音频 MCD 的组成模块

图 2-37 表示波形音频媒体控制驱动程序(AUDIOMCT.DLL)的大致轮廓。

图 2-37 波形音频 MCD 模块

2.7.4.1 波形音频 MCD DLL(AUDIOMCT.DLL)

表 2-13 描述了 AUDIOMCT.DLL 所处理的文件及消息。

表 2-13 波形音频 MCD 文件的说明

文件	说 明
ADMCOPEN.C	处理 MCI_OPEN 消息。当收到 MCI_OPEN 消息,一个流 MCD 将执行以下动作: 1.检查标志及使指针有效; 2.从 INI 文件中获取默认值(如有必要的话); 3.处理 MCI_OPEN_PLAYLIST(如果支持的话); 4.处理 MCI_OPEN_MMIO(如果支持的话); 5.处理 MCI_OPEN_ELEMENT(如果支持的话); 6.与混响放大器连接; 7.获取流协议关键字(如有必要的话)。
LOADSUBS.C	提供 MCI_OPEN 和 MCI_LOAD 使用的工具函数,包括: 1.创建临时文件名(CheckForValidElement); 2.放弃过程命令(Load AbortNotify); 3.处理 OPEN_MMIO 标志(OpenHandle); 4.判别在录制或播放模式中何时打开卡片(OpenHandle)(ProcessElement); 5.处理临时文件(ProcessElement); 6.用 MMIO 打开一个文件(ProcessElement); 7.处理 MCI_READ_ONLY 标志(ProcessElement); 8.创建临时文件(SetupTempFiles); 9.利用 mmioSendMessage API 与 I/O 过程对话(SetupTempFiles); 10.获取一个连接并打开连接的设备; 11.设置流处理器(StreamSetup); 12.处理 MCI_NOTIFY 或 MCI_WAIT 及回调句柄(NotifyWaitSetup)。
AUDIOMCD.C	处理 MCIDRV_RESTORE 消息。当一个流 MCD 重新获得对所附属于的设备的控制时,它将收到一条恢复(restore)消息(例如当某人退出某个其它的应用程序而导致我们获得对这个设备的使用权的时候)。在收到恢复消息后,MCD 要检查是否处于暂停状态。如果是的话,则恢复启动数据流。 处理 MCIDRV_SAVE 消息。当一个流 MCD 失去对所附属于的设备的控制时,它将收到一条保存(save)消息(例如当某人启动了某个其它的应用程序而夺走了波形音频设备的使用权的时候)。在收到保存消息后,MCD 要检查当前是否在传递数据(录制或播放)。如果是的话,则设置一个标志说明已保存了 MCD。
ADMCLOAD.C	处理 MCI_LOAD 消息。本文件阐述了以下概念: 1.收到 MCI_LOAD 后怎样检查标志; 2.怎样中止一个命令处理; 3.收到 MCI_LOAD 后为何要去掉提示点(cuepoint)/位置报告(positionadvise);

文件	说　明
	4 .收到 MCI. LOAD 后怎样处理 OPEN. MMIO; 5 .为什么在收到 MCI. LOAD 后重关联数据流是可行的,以及何时是适合这样做的。
ADMCCAP .C	处理 MCI. GETDEVCAPS 消息。本文件阐述了用于处理 MCI. GETDEVCAPS 的几个概念: 1 .怎样处理该消息命令。本 MCD 支持诸如 play,close 这样的消息,而不支持诸如 sysinfo 这样的消息(或对调用者来说则是"命令")。 2 .怎样处理各项能力(capability)的标志,它们描述了某具体专项的能力(比如录制的能力)。MCD 对各专项可能支持,也可能不支持。
ADMCSTAT .C	处理 MCI. STATUS 消息。本文件阐述了用于处理 MCI. STATUS 的几个概念,包括: 1 .何时及是否通知媒体在数据流中达到的位置; 2 .怎样确定已存在的文件的长度; 3 .怎样确定正在录制的文件的长度; 4 .与混响放大器通信以确定音量,及其它放大器特有命令; 5 .通知我们的当前模式; 6 .通知当前的时间格式。
ADMCCUE .C	处理 MCI. CUE 消息。应用程序通常调用 MCI. CUE 以缩短启动录制或播放所需的时间。对于一个流 MCD, MCI. CUE 被翻译成为 SPI. START. PREROLL。预处理(preroll)将填充所有的初始流缓冲区,但并不启动数据流。然后,MCI. PLAY 或 MCI. RECORD 调用 SpiStartStream 函数,数据流就可立即被启动。 收到 MCI. CUE 消息后,一个流 MCD 要执行以下的动作: 1 .检查标志并使指针有效; 2 .检查数据流是否已经处于提示(cue)状态。如果是的话,则返回成功信息; 3 .如果调用者要为输出做提示(cue),执行下面步骤: (1) 停止一切命令处理; (2) 如果数据流的方向有误(即,在录制状态)则销毁它; (3) 创建数据流(如有必要的话); (4) 预处理(preroll)数据流。 4 .如果调用者要为输入做提示,执行以下步骤: (1) 停止一切命令处理; (2) 如果数据流的方向有误,则销毁它; (3) 创建数据流(如有必要的话); (4) 预处理数据流。 处理 MCI. SET. CUEPOINT 消息。收到该消息后,一个流 MCD 要执行以下动作: 1 .检查标志并使指针有效; 2 .使提示点(cuepoint)有效或无效(取决于传来的是哪个标志)。
ADMCSEEK .C	处理 MCI. SEEK 消息。收到该消息后,一个流 MCD 要执行以下动作:

续表

文件	说　　明
	1 .确认传来的标志是有效的; 2 .如果传来了 MCI_FROM 和 MCI_TO 参数,核查之; 3 .确认所有传来的指针都是有效的; 4 .停止在另一个线程中活动的任何命令; 5 .如果不存在数据流,就创建一个; 6 .如果先前已创建了一个数据流,保证它处于停止状态。
ADMCPLAY.C	处理 MCI_PLAY 消息。当收到该消息后,一个流 MCD 要执行以下动作: 1 .总是先检查标志并使内存有效。这样,若标志无效,则前一个命令不会被打断。 2 .如果在另一个线程中有正在活动的命令(即,正在执行播放,录制或保存),则或者废弃(对于录制或保存命令)或者更新(对于播放命令)它。这就要停止这个流并向调用者发送一条消息。 3 .若数据流的方向有误(比如设在录制状态),则销毁它。 4 .如果不存在数据流,就创建一个。如果流处理器需要关联一个数据对象,就在此时做此事。 5 .如果我们在播放数据流之前销毁了一个录制数据流,则须确保播放数据流与先前的录制数据流具有相同的位置。 6. 允许任意的事件(比如提示点或位置报告)。 7. 启动数据流。 8. 等待数据流事件。 9. 如果有必要,停止数据流。 10. 如果使用了 MCI_NOTIFY,则通知调用者命令已完成。
ADMCRECD.C	处理 MCI_RECORD 消息。收到该消息后,一个流 MCD 要执行以下动作: 1 .总是先检查标志并使内存有效,这样,若标志无效,则前一个命令不会被打断。 2 .如果在另一个线程中有正在活动的命令(即正在执行播放,录制或保存),则或废弃(对播放或保存命令)或更新(对录制命令)它,这就要停止这个流并向调用者发送一条消息。 3 .如果数据流的方向有误(比如设在播放状态),则销毁它。 4 .如果不存在数据流,就创建一个。如果流处理器需要关联一个数据对象就在这时做此事。 5 .如果我们在创建录制数据流之前销毁了一个播放数据流,则要在录制数据流中找到与先前的播放数据流相同的位置。 6 .允许任意的事件(比如提示点或位置报告)。 7 .启动数据流。 8 .等待数据流事件。 9 .如有必要,停止数据流。 10 .如果使用了 MCI_NOTIFY,则通知调用者命令已完成。
ADMCCLOS.C	处理 MCI_CLOSE 消息。收到该消息后,一个流 MCD 要执行下列动作: 1 .停止所有在其它线程活动的命令; 2 .销毁所有活动的数据流;

续表

文件	说　　明
	3 . 关闭所有打开的文件; 4 . 删除任何临时文件; 5 . 关闭任何连接的设备(比如混响放大器) ; 6 . 如果使用了 MCI. NOTIFY,通知调用者命令已完成。
ADMCCONN .C	处理 MCI. CONNECTOR 消息。本源文件阐述了怎样允许、禁止和查询连接符,以及怎样向一个已连接的 MCD 传送消息。
ADMCREST .C	处理 MCI. STOP 消息。收到该消息后,一个流 MCD 要执行以下动作: 1 . 确认传来的标志是有效的; 2 . 停止所有在其它线程活动的命令; 3 . 如果先前已创建了一个数据流,确保它处在停止状态; 4 . 如果数据流处在暂停状态,则用 STOP. PAUSE 以确保无数据丢失。 处理 MCI. RESUME 消息。收到该消息后,一个流 MCD 要执行以下动作: 1 . 确保无标志传来; 2 . 若数据流处于暂停状态,则恢复启动它。 处理 MCI. SETPOSITIONADVISE 消息。对一个流 MCD 来说,一个位置报告(positionadvise)只不过是每 x 单位时间一次的提示点(cuepoint)。为了允许位置报告,要做以下几步: 1 . 检查标志并使指针有效; 2 . 如果调用者要求允许位置报告,那么: (1) 若已创建了数据流,则允许重复发生的提示点事件; (2) 否则,设置一个标志,待以后再允许该事件。 处理 MCI. PAUSE 消息。收到该消息后,一个流 MCD 要执行以下动作: 1 . 确认无标志传来; 2 . 暂停数据流的传递; 3 . 设置标志表明已在暂停状态。
ADMSAVE .C	处理 MCI. SAVE 消息。MCI. SAVE 不是一个必需命令,但是若一个流 MCD 要使用临时文件则需用这命令: 1 . 确认标志有效; 2 . 确认所有指针指向有效的内存单元; 3 . 废止一切活动中的操作; 4 . 最后,进行保存(save)操作。
AUDIOSUB .C	包含下列工具函数: 1 . 通知修正处理; 2 . 处理除 MCI. WAIT 和 MCI. NOTIFY 之外的调用(PostMDMMessage) ; 3 . 使用 mmioGetHeader 从文件中获取音频设置(GetHeader) ; 4 . 创建一个播放节目表(playlist)数据流(Associate Playlist) ; 5 . 安装 I/ O 过程(InstallIOProc) ; 6 . 利用不同功能与 I/ O 过程通信; 7 . 用各种文件格式处理声音文件(OpenFile) ; 8 . 时间格式转换(ConvertToMM + ConvertTimeFormat) ; 9 . 创建一个 SPI 流(CreateNAssociateStream) ;

续表

文件	说　　明
	10．关联一个 SPI 流(CreateNAssociateStream)； 11．为在 play to/ record to 中设置一个事件(DoTill Event)。
ADMCPST.C	处理 MCI_PASTE 消息。本文件利用 MMIO 函数和剪贴板函数: 1．处理 MCI_FROM_BUFFER 或 MCI_TO_BUFFER 标志; 2．若无上述二标志传来,则处理默认的定位; 3．用 MMIO 向文件中插入信息。
ADMCCOPY.C	处理 MCI_COPY, MCI_DELETE, MCI_CUT, MCI_UNDO 和 MCI_REDO 消息。这是一个通用的例程,用来把信息放入剪贴板: 1．默认地为 cut/ copy/ delete 消息定位; 2．使用 MMIOM_DELETE 从文件删除信息; 3．在 cut/ copy/ delete 之后定位; 4．使用 MMIOM_UNDO 和 MMIOM_REDO 消息。
ADMCEDIT.C	包含剪贴板函数所用的工具(如 cut 和 copy),本文件阐述了: 1．怎样在编辑操作中处理 MCI_FROM/ MCI_TO(CheckEditFlags); 2．确认默认的 from/ to 位置; 3．通用的废止流的例程; 4．从剪贴板获得信息; 5．把信息放入剪贴板; 6．用 MMIO 内存文件处理剪贴板数据。
ADMCINI.C	阐述怎样分析出 INI 文件中的设备特有参数。
STRMSUBS.C	包含 play, record 和 cue 所用的流例程: 1．怎样关闭 mmio 文件和临时文件(CloseFile); 2．怎样停止一个录制/ 播放流(StopStream); 3．怎样通过使用一个信号来保护命令过程不会在敏感部位被废止; 4．为什么在 MCI_SET 后要销毁数据流(DestroySetStream); 5．怎样更换和废止通知; 6．为什么和怎样为 MCI_TO 设置提示时刻暂停事件(cue time pause event)(ProcessFromToFlags); 7．怎样允许提示点和位置更改(EnableEvents); 8．网络功能(BeginQualityofService, EndQualityofService)。

2.8　控制非流式设备:CD 音频 MCD

CD 音频 MCD 控制一种非流式设备:CD-ROM 驱动器。因为 CD_ROM 驱动器是非流式设备,所以它不需要 SSM 提供的带缓冲区的 I/ O。它靠内部的数模转换器(DAC)来播放声音。这是与声卡功用相当的非流式设备。这样,CD 音频 MCD 必须使用 IOCtl 接口来处理它自己的命令,以在设备内部处理数据流。

组成 CD 音频 MCD 的模块如图 2-38 所示。

图 2-38　CD 音频媒体控制驱动程序模块

2.8.1　设置音频属性

因为 CD-ROM 驱动器是非流式设备。所以它没有链接到混响放大器。这意味着驱动程序必须自己处理 MCI_SET 命令,通过 IOCtl 接口来设置音频属性(如音量改变等)。

2.8.2　处理 MCI_PLAY 命令

CD 音频 MCD 通过 mciDriverEntry 接收命令,这与波形音频 MCD 相同。命令被送往售主特定驱动程序(vendor-specific driver,简写为 VSD)。

让我们跟踪 MCI_PLAY 命令的处理过程,来看看 CD 音频 MCD 怎样处理命令。一开始,函数 vsdDriverEntry 调用函数 process_msg,传递消息和参数,如图 2-39 所示。

```
ULONG APIENTRY vsdDriverEntry(PVOID pInstance,USHORT usMessage,
                              ULONG *pulParam1,PVOID pParam2,
                              USHORT usUserParm)
{
  ULONG rc, ulPlTemp = MCI_WAIT;
  USHORT try = 1;
  if (pulParam1 == 0L)
     pulParam1 = &ulPlTemp;
  /*check to see if the drive is open, unless it is an Open message*/
  if (usMessage == MCI_OPEN)
     rc = CDAudOpen(*pulParam1,(MMDRV_OPEN_PARMS*)pParam2);
  else
  {
```

```
/ * if the device is closed try reopening it unless you are closing */
if (((PINST)lpInstance) - > hDrive = = 0 & & usMessage ! = MCI_CLOSE)
{
  rc = CD01_Open((PINST)lpInstance);
  / * Clear commands not needing an open hardware device */
  if (rc = = MCIERR_DEVICE_NOT_READY)
    if ((usMessage = = MCI_DEVICESETTINGS)||
      (usMessage = = MCI_GETDEVCAPS)||
      (usMessage = = MCI_INFO)||
      (usMessage = = MCIDRV_REGISTER_DRIVE)||
      (usMessage = = MCI_SET_CUEPOINT)||
      (usMessage = = MCI_SET_POSITION_ADVISE)||
      (usMessage = = MCIDRV_CD_STATUS_CVOL)||
      (usMessage = = MCIDRV_SYNC & &
        ! ( * pulParam1 & MCIDRV_SYNC_REC_PULSE)))
      rc = MCIERR_SUCCESS;

  }/ * of if drive needs to be open */
  else    / * drive was opened */
    rc = MCIERR_SUCCESS;

  if ( !rc)
    do
    {
      / * process message */
      rc = process_msg((PINST)lpInstance,usMessage,
                        pulParam1,pParam2,usUserParm);
    if(rc = = MCIERR_DEVICE_NOT_READY||     / * ERROR RECOVERY */
       rc = = MCIERR_MEDIA_CHANGED)
    {
      if (((PINST)lpInstance) - > Drive = = 0)/ * drive is closed */
      {                                  / * do not reissue commands */
        rc = MCIERR_SUCCESS;
        break;
      }
      else
        if (try = = 2)
          break;                         / * quit after 2 tries .*/
      else
      {
        rc = CDAudErrRecov((PINST)lpInstance);
        if (rc)                          / * error is still there, exit */
          break;
        else
```

```
                try++;
            }/ * of else only tried the command once (try==1) */
        }/ * of if the drive was not ready */
        else
            break;                          / * clear flag to exit */
      } while (try);  / * end of do loop and if no open error */
    }/ * of else command was not MCI_OPEN */

    return(rc);

}/ * of vsdDriverEntry() */
```

图 2-39 vsdDriveEntry 函数(IBMCDROM.C)

process-msg 例程有一个很长的分支语句、我们感兴趣的是 MCI_PLAY 的这个分支(case)当 process-msg 调用 CD01_Play 时,它传递了以下信息:

· Instance

· msgParam1

· 从 msgParam2 得到的 FROM 的值

· 从 msgParam2 得到的 TO 的值

· MCI_PLAY 命令

```
static ULONG process_msg (PINST pInst,USHORT usMessage,
                          ULONG * pulParam1,PVOID pParam2,USHORT usUserParm)
{
   ULONG rc;

   DosRequestMutexSem(pInst->hInstSem,WAIT_FOREVER);
   if (usMessage != MCI_PLAY)
     DOSReleaseMutexSem(pInst->hInstSem);    // No protection needed

/ * process message */
switch(usMessage)

{
  case MCI_CLOSE:
     rc = CDAudClose(pInst, * pulParam1);
     break;
  case MCI_CUE:                              / * Pre-roll */
     rc = CD01_Cue(pInst);
     break;
  case MCI_GETDEVCAPS:                       / * Get Device Capabilities */
     rc = CD01_GetCaps( * pulParam1,(MCI_GETDEVCAPS_PARMS * )pParam2);
     break;
  case MCI_GETTOC:                           / * Get Table of Contents */
```

```
      if (*pulParam1 & WAIT_NOTIFY_MASK)
         rc = MCIERR_INVALID_FLAG;
      else
         rc = CD01_GetTOC(pInst,(MCI_TOC_PARMS *)pParam2);
      break;
   case MCI_INFO:
      rc = CDAudInfo(pInst, *pulParam1,(MCI_INFO_PARMS *)pParam2);
      break;
/* case MCI_OPEN:     open was already done in vsdDriverEntry() */

   case MCI_PAUSE:
      rc = CD01_Stop(pInst,TIMER_PLAY_SUSPEND);
      break;
   case MCI_PLAY:
      rc = CD01_Play(pInst, pulParam1,
                  ((MCI_PLAY_PARMS *)pParam2)->ulFrom,
                  ((MCI_PLAY_PARMS *)pParam2)->ulTo,usUserParm,
                  ((MCI_PLAY_PARMS *)pParam2)->hwndCallback);
      break;
   case MCIDRV_REGISTER_DISC:                    /* Register Disc */
      rc = CDAudRegDisc(pInst,REG_BOTH,(MCI_CD_REGDISC_PARMS *)
         pParam2);
      break;
   case MCIDRV_REGISTER_DRIVE:                   /* Register Drive */
      rc = CDAudRegDrive(pInst,(MCI_CD_REGDRIVE_PARMS *)pParam2);
      break;
   case MCIDRV_REGISTER_TRACKS:                  /* Register Tracks */
      rc = CD01_RegTracks(pInst,(MCI_CD_REGTRACKS_PARMS *)pParam2);
      break;
   case MCIDRV_RESTORE:
      rc = CD01_Restore(pInst,(MCIDRV_CD_SAVE_PARMS *)pParam2);
      break;                                     /* Unpause */
   case MCI_RESUME:
      rc = CD01_Resume(pInst);
      break;
   case MCIDRV_SAVE:
      rc = CD01_Save(pInst,(MCIDRV_CD_SAVE_PARMS *)pParam2);
      break;
   case MCI_SEEK:
      rc = CD01_Seek(pInst,((MCI_SEEK_PARMS *)pParam2)->ulTo);
      break;
```

```
    case MCI_SET:
        rc = CDAudSet(pInst,pulParam1,(MCI_SET_PARMS * )pParam2);
        break;
    case MCI_SET_CUEPOINT:
        rc = CD01_CuePoint(pInst, * pulParam1,(MCI_CUEPOINT_PARMS * )
            pParam2);
        break;
    case MCI_SET_POSITION_ADVISE:
        rc = CD01_PosAdvise(pInst, * pulParam1,(MCI_POSITION_PARMS * )
            pParam2);
        break;
    case MCIDRV_CD_SET_VERIFY:
        rc = CDAudSetVerify( * pulParam1);
        break;
    case MCI_STATUS:
        rc = CDAudStatus(pInst, * pulParam1,(MCI_STATUS_PARMS * )pParam2);
        break;
    case MCIDRV_CD_STATUS_CVOL:
        rc = CDAudStatCVol( &((MCI_STATUS_PARMS * )pParam2) - > ulReturn);
        break;
    case MCI_STOP:
        rc = CD01_Stop(pInst,TIMER_EXIT_ABORTED);
        break;
    case MCIDRV_SYNC:
        rc = CD01_Sync(pInst, * pulParam1,(MCIDRV_SYNC_PARMS * )pParam2);
        break;
   / * List unsupported functions */
    case MCI_ACQUIREDEVICE:   case MCI_CONNECTION:   case MCI_CONNECTOR:
    case MCI_CONNECTORINFO:   case MCI_DEVICESETTINGS:
    case MCI_DEFAULT_CONNECTION:                     case MCI_ESCAPE:
    case MCI_LOAD:            case MCI_MASTERAUDIO:  case MCI_RECORD:
    case MCI_RELEASEDEVICE:   case MCI_SAVE:         case MCI_SPIN:
    case MCI_STEP:            case MCI_SYSINFO:      case MCI_UPDATE:
    case MCIDRV_CD_READ_LONG:
        rc = MCIERR_UNSUPPORTED_FUNCTION;
        break;
    default:rc = MCIERR_UNRECOGNIZED_COMMAND;
  }/ * of switch */
  return(rc);
```

图 2-40 Process-msg 子例程(IBMCDROM .C)

当 CD01_Play 收到 PLAY 命令,程序跳到调用 callIOCtl 的那一行。

```
ULONG CD01_Play(PINST pInst,ULONG *pulParam1,ULONG ulFrom, ULONG ulTo,
                USHORT usUserParm,HWND hwndCallback)
{
   ULONG rc;
   ULONG ulThreadID;
   ULONG cnt;
  /* Stop drive before issuing next play command */
   if ((pInst->usPlayFlag==TIMER_PLAYING)||
        (pInst->usPlayFlag==TIMER_PLAY_SUSPEND)||
        (pInst->usPlayFlag==TIMER_PLAY_SUSP_2))
      if (*pulParam1 & MCI_NOTIFY)
        CD01_Stop(pInst,TIMER_EXIT_SUPER);
      else
         CD01_Stop(pInst,TIMER_EXIT_ABORTED);

     /* prepare for play call */
      pInst->ulCurPos=ulFrom;
      pInst->ulEndPos=ulTo;
      pInst->usPlayNotify=(USHORT)(*pulParam1&(MCI_WAIT|MCI_NOTIFY));
      if (*pulParam1 & MCI_NOTIFY)
      {
         pInst->usPlayUserParm=usUserParm;
         pInst->hwndPlayCallback=hwndCallback;
         *pulParam1==MCI_NOTIFY;
     }/ *notify flag was used */

     if (*pulParam1 & MCI_WAIT)
        rc = CD01_Timer(pInst);     /* returns when play commands end */
     else
     {
        DosResetEventSem(pInst->hReturnSem,& cnt);  /* force a wait
        rc=DosCreateThread(& ulThreadID,(PFNTHREAD)CD01_Timer,
                                  (ULONG)pInst,0L,THREAD_STACK_SZ);
        if (rc)
        {
            rc=MCIERR_OUT_OF_MEMORY;
            DosPostEventSem(pInst->hReturnSem);
   }
        else/ *wait for new thread to process enough */
        {
            /* Let MCD know not to send notification by returning wait */
             *pulParam1=(*pulParam1& MCI_NOTIFY)|MCI_WAIT;

            /* wait for new thread to process enough */
```

```
        DosWaitEventSem(pInst->hReturnSem,WAIT_FOREVER);
      }
      DosReleaseMutexSem(pInst->hInstSem);
   } /* else no wait flag was used */

   return(rc);
} /* of CD01_Play() */
/****************************************************************/
/*                                                              */
/* SUBROUTINE NAME: CD01_PlayCont                               */
/*                                                              */
/* DESCRIPTIVE NAME: CD Play Continue.                          */
/*                                                              */
/* FUNCTION: Continue to play audio data to internal DAC(s) from a */
/*           MCIDRV_RESTORE or MCIDRV_SYNC command.             */
/*                                                              */
/* PARAMETERS:                                                  */
/*      PINST pInst        --Instance pointer.                  */
/*      ULONG ulFrom       --From address.                      */
/*      ULONG ulTo         --To address in MMTIME.              */
/*                                                              */
/* EXIT CODES:                                                  */
/*      MCIERR_SUCCESS     --action completed without error.    */
/*      MCIERR_DEVICE_NOT_READY  --device was not ready,no disc. */
/*      MCIERR_MEDIA_CHANGED     --Disc changed.                */
/*                                                              */
/* NOTES:                                                       */
/*                                                              */
/****************************************************************/
ULONG CD01_PlayCont(PINST pInst,ULONG ulFrom,ULONG ulTo)
{
   ULONG rc;
   BYTE param[PLAYAUD_PMAX] = {'C','D','0','1',RBMODE};
   ULONG ulDataLen = STANDRD_DMAX,ulParamLen = PLAYAUD_PMAX;

   /* convert starting MM Time into Redbook 2 format */
    *(ULONG *)&param[STARTFFFLD] = REDBOOK2FROMMM(ulFrom);

   /* convert ending MM Time into Redbook 2 format */
    *(ULONG *)&param[END_FF_FLD] = REDBOOK2FROMMM(ulTo);

   /* Stop drive before issuing next play command */
   CD01_Stop(pInst,TIMER_PLAY_SUSPEND);

   /* play drive */
```

```
        rc = CallIOCtl(pInst,CDAUDIO_CAT,PLAY_AUDIO,
                       param,ulParamLen,&ulParamLen,
                       NULL,ulDataLen,&ulDataLen);
        if (!rc)
           pInst->ulCurPos = ulFrom;

      /* if Timer was stopped, continue timer loop */
      if (pInst->usPlayFlag == TIMER_PLAY_SUSPEND)
          pInst->usPlayFlag = TIMER_PLAYING;
      DosPostEventSem(pInst->hTimeLoopSem);

    return(rc);
}/* of CD01_PlayCont() */
```

图 2-41　CD01_Play 和 CD01_PlayCont 子例程(IBMCDPRO.C)

CallIOCtl 例程发出 DosDevIOCtl 请求,传递以下参数:hDrive 是从 DosOpen 调用得到的设备句柄,ulCat 是 IOCtl 的类别,ulFunction 是功能号。此外还有缓冲区指针、缓冲区大小和长度以及指向参数和数据缓冲区长度的指针。参数缓冲区用于输入,数据缓冲区用于输出。指向缓冲区长度的指针使物理设备驱动程序能够返回缓冲区的实际长度。

```
ULONG CallIOCtl(PINST pInst, ULONG ulCat,ULONG ulFunction,
                PVOID pParam,ULONG ulPLen,ULONG *pulPLen,
                PVOID pData, ULONG ulDLen,ULONG *pulDLen)
{
  ULONG rc;

  DosRequestMutexSem(pInst->hIOSem,(ULONG)-1L);
  rc = DosDevIOCtl(pInst->hDrive,ulCat,ulFunction,
                   pParam,ulPLen,pulPLen,
                   pData,ulDLen,pulDLen);
  DosReleaseMutexSem(pInst->hIOSem);

  switch(rc)
  {
     case 0L:
        rc = MCIERR_SUCCESS;
        break;
    case 0xFF03:
        rc = MCIERR_UNSUPPORTED_FUNCTION;
        break;
    case 0xFF06:
    case 0xFF08:
       rc = MCIERR_OUTOFRANGE;
       break;
    case 0xFF04:
    case 0xFF0C:
```

```
        rc = MCIERR_HARDWARE;
        break;
     case 0xFF10:
        rc = MCIERR_MEDIA_CHANGED;
        break;
     default:
        rc = MCIERR_DEVICE_NOT_READY;
     }
     return(rc);
}/ * of CallIOCtl() */
```

图 2-42 CallOCtl 子例程(IBMCDMSC.C)

2.8.3 CD 音频 MCD 的组成模块

图 2-43 显示了 CD 音频 MCD 的大致轮廓以及 IBM 3510—001 型 CD-ROM 驱动器的 VSD(IBMCDRT.DLL)。

图 2-43 CD 媒体控制驱动程序模块

2.8.3.1 CD 音频媒体控制驱动程序(CDAUDIOT.DLL)

这个 CD 音频 MCD(CDAUDIOT)处理从 MDM 授收的命令。若它不能完全处理某命令,就把它发给 VSD 中的硬件专用码。

下面详细介绍了 CDAUDIOT.DLL 中的文件和例程。

CDAUDIO.C 文件:本模块包含了 DLL 的入口点。它根据请求的命令消息,使全局变量有效并核查参数。如果基本核查无误,就调用一个特殊的处理函数。如表 2-14 示。

CDAUDPRO.C 文件:本模块包含了用于处理命令消息的与硬件无关的代码。有些命令可以完全由 MCD 处理,而还有另一些则需要 VSD 与硬件打交道或了解具体的 VSD 驱动器的特有信息。如表 2-15 示。

表 2-14 CDAUDIO.C 的过程

过程	说　明
mciDriverEntry	指明从 MDM 到 MCD 的入口点。
pre_process_msg	使设备准备好处理命令。
process_msg	处理请求的命令消息。
verify_entry	核实入口参数为有效。
QMAudio	查询主音频的当前设置。
Register	注册设备。
ReRegister	注册盘片和轨道。
VSDReturn	指明从 VSD 到 MCD 的入口点,处理来自 VSD 的返回信息。
SetTrackInst	设定实例中的轨道信息。
ValPointer	使欲记录结构的地址有效。

表 2-15 CDAUDPRO.C 的过程

过程	说　明
ProcClse	处理 MCI_CLOSE 命令。
ProcConnector	处理 MCI_CONNECTOR 命令。
ProcCue	处理 MCI_CUE 命令。
ProcCuePoint	处理 MCI_SET_CUEPOINT 命令
ProcGeneral	处理穿行的 MCI 命令。
ProcCaps	处理 MCI_GETDEVCAPS 命令。
ProcInfo	处理 MCI_INFO 命令。
ProcMAudio	处理 MCI_MASTERAUDIO 命令。
ProcOpen	处理 MCI_OPEN 命令。
ProcPause	处理 MCI_PAUSE 命令。
ProcPlay	处理 MCI_PLAY 命令。
ProcPosAdvise	处理 MCI_SET_POSITION_ADVISE 命令。
ProcRestore	处理 MCIDRV_RESTORE 命令。
ProcResume	处理 MCI_RESUME 命令。
ProcSave	处理 MCIDRV_SAVE 命令。
ProcSeek	处理 MCI_SEEK 命令。

续表

过程	说　　明
ProcSet	处理 MCI_SET 命令。
ProcSetSync	处理 MCI_SET_SYNC_OFFSET 命令。
ProcStatus	处理 MCI_STATUS 命令。
ProcStop	处理 MCI_STOP 命令。
ProcSync	处理 MCIDRV_SYNC 命令。

CDAUDUTL.C 文件:本模块包含增补 CDAUDPRO.C **中的处理命令的与硬件无关的代码,它也包含了工具函数。如表** 2-16 **示。**

表 2-16　CDAUDUTL.C 的过程

过程	说　　明
SetAudio	用 MCI_SET 设定音频信息。
SetConnector	允许或禁止一个关联。
SetCuePoint	允许提示点。
StatusMCD	从 MCD 信息获取状态。
StatusMCDDef	从 MCD 默认信息获取状态。
DisableEvents	禁止提示点和位置报告。
GetTimeAddr	将时间格式转换为 MMTIME,或反之。
GetTimeAddrRC	把返回码调整为时间格式。
GetTrackInfo	获取某特定轨道的信息。
ValAddress	使地址在范围内,使有效。
ValState	使逻辑设备的状态有效。
vsdResponse	处理 VSD 的响应。

CDMCINIT.C 文件:本模块初始化可重入的 DLL。**如表** 2-17 **示。**

表 2-17　CDMCINIT.C 的过程

过程	说　　明
_DLL_InitTerm	指明 OS/2 装入程序的入口点。
CDMCInitialization	获取初始化堆。
CDMC_Exit	结束后清除实例。

CDMCCOMN.C 文件:本模块包含 CD MCD **与** IBM CD-ROM **驱动器的** VSD **间公用的函数。如表** 2-18 **示。**

表 2-18　CDMCCOMN.C 的过程

过程	说明
parse_DevParm	分析设备特有参数。
get_token	取得下一个标记(token),若是 null 则终止。

HHPHEAP.C 文件:本模块包含公用的堆管理函数。如表 2-19 **示。**

表 2-19　HHPHEAP .C 的过程

过程	说　明
HhpCreateHeap	创建第一个堆。
NewHeap	创建一个新的堆(一段)。
HhpAllocMem	分配一些内存。
LocateFreeMem	定位一个自由内存块。
HhpFreeMem	释放内存。
CollapseFreeBlock	去除一个堆内的残片。
CollapseFreeHeap	去除多个堆中间的残片。
HhpDestroyHeap	销毁一个堆。
HhpAllocBuffer	从系统分配一个缓冲区。
HhpFreeBuffer	释放已分配的缓冲区。
HhpAccessBuffer	设置一个可访问的段。
HhpAccessHeap	访问一个共享的堆。
HhpReleaseHeap	释放一个堆。
AddPid	向 PID 表中增加一个过程标识符。
ReallocHeap	重新为一个堆分配内存。
HhpGetPID	获取堆的 PID。
HhpDumpHeap	把堆的内容转移到标准输出。

2.8.3.2　CD-ROM 驱动器的 VSD(IBMCDRT .DLL)

下面介绍了 IBMCDRT .DLL 中的文件和例程。如表 2-20 示。

· CDMCINIT .C 文件　见表 2-17;

· CDMCCOMN .C 文件　见表 2-18;

· HHPHEAP .C 文件　见表 2-19;

· IBMCDPRO .C 文件　本模块包含 DLL 的入口点。它包含特定的 CD-ROM 驱动器的设备相关的代码。

表 2-20　IBMCDROM .C 的过程

过程	说　明
vsdDriverEntry	指明从 MCD 到 VSD 的入口点。
process-msg	处理请求的命令消息。
CDAudClose	关闭一个实例。
CDAudErrRecov	错误恢复例程。
CDAudInfo	返回元件的信息。
CDAudOpen	打开一个实例。
CDAudRegDisc	为逻辑设备注册一个盘片。
CDAudRegDrive	为发出调用的 MCD 注册一个驱动器。
CDAudset	设置设备的各种属性。
CDAudSetVerify	检测 SET 命令的标志。
CDAudStatus	返回要求的属性。
CDAudStatCVol	返回映象元件的音量值。

IBMCDPRO .C 文件:本模块处理硬件请求。如表 2-21 示。

表 2-21　IBMCDPRO.C 的过程

过程	说　　明
CD01-Cue	预处理一个驱动器。
CD01-CuePoint	设置一个提示点(cuepoint)。
CD01-GetCaps	获取设备的能力。
CD01-GetDiscInfo	获取盘片的状态信息。
CD01-GetID	获取盘片的 CD ID 号。
CD01-GetPosition	获取扫描头的位置。
CD01-GetState	获取设备的状态。
CD01-GetTOC	返回内容表(MMTOC 形式)。
CD01-GetVolume	获取设备的音量设置。
CD01-LockDoor	锁住或开启驱动器的门。
CD01-Open	打开特定的设备或驱动器。
CD01-Play	初始化播放操作。
CD01-PlayCont	继续播放操作。
CD01-PosAdvise	设立一个位置报告命令。
CD01-RegTracks	在盘上注册轨道。
CD01-Restore	恢复保存的实例。
CD01-Resume	从暂停状态恢复到播放状态。
CD01-Save	保存当前的实例。
CD01-Seek	查找一个特定的红皮书地址。
CD01-SetVolume	设置驱动器的音量。
CD01-Stop	停止 CD 的播放。

IBMCDMSC.C 文件:本模块处理杂项函数,如播放命令的定时器例程,确认驱动器,及封装 IOCtl 调用。如表 2-22 示。

表 2-22　IBMCDMSC.C 的过程

过程	说　　明
CD01-StartPlay	开始播放。
CD01-Sync	与 MDM 请求同步。
CD01-Timer	播放操作的定时器例程。
CD01-Timernotify	设置/ 通知事件的定时器例程。
GetTableName	获取 CD 表的全路径名称。
OpenFirstTime	首次设备打开检测。
QueryTable	查询 CD 查询表(look-up table)。
CallIOCtl	用 IOCtl 调用硬件。

2.9　资源单元和资源类

MDM 在管理设备上下文时,使用了这样一些抽象的概念:资源单元、资源类及有效类组合。

资源单元为资源管理程序提供了这样一个尺度,以确定在某给定时间可以有多少个活动的设备上下文。每个设备都给出它能够同时处理的资源单元的数目。

除了总的资源数,每个资源类还有一个可用的资源单元数的上限。这使得 MDM 可以确定从某特定的类中可以同时激活的设备上下文个数。它的最大值是在安装过程中提供的。

在设备上下文的 MCI_OPEN 命令下,设备上下文所需的资源单元及资源类是在 MMDRV_OPEN_PARMS 结构中返回的。若用 MCIDRV_CHANGERESOURCE 消息调用 MDM,则 MDM 就可以更改设备上下文所需的资源单元和资源类。例如,一个波形音频设备为单声波形分配一个资源单元,为立体声波形分配两个资源单元,而一个加载命令可以更改该设备上下文所需的资源单元。

资源管理的最后一个构件是在安装过程中提供的,这就是有效类组合。一个设备可以拥有多个资源类,但可能不允许某几个类同时拥有活动的设备上下文。下面是一个 ProAudioSpectrum 16** 卡的例子。它最多可有两个可用的资源单元。它使用两个类,一个用于波形音频,另一个用于 MIDI。对于每个类来说,都最多只有一个资源单元可供活动的设备上下文使用。它可以有任意多个非活动的设备上下文。最后,两个类可以同时各拥有一个活动的设备上下文。这就是说,这个卡可以支持一个波形音频及一个 MIDI,或二者之一。

2.10 向多媒体设置笔记本中插入页

多媒体设置笔记本是一个程序,它提供了一个用户接口,以管理由媒体设备管理器(MDM)注册的多媒体设备的特性。本节阐述了怎样在多媒体设置笔记本中插入设置页。

有两种办法可以向多媒体设置笔记本中插入设置页。下面我们将集中讨论第一种办法,即使用 MCI_DEVICESETTINGS 消息为特定的 MCD 插入一页。第二种办法将在第5章“安装要求”的 5.3.3.3 节“定义其它 INI 文件更改信息”中讨论。该办法引入了一种基于设备类型(而不是 MCD)的多媒体设置的注册机制,来插入设置页。例如,那些应用于系统或某特定类的所有媒体控制接口设备的页。

注意:

1. 向一个设备对象的多媒体设置笔记本中插入一页所用的数据结构,定义在头文件 MMSYSTEM.C 中。

2. 关于笔记本控制窗口处理过程的进一步资料,参阅《OS/2 PM 参考》一书。

如果一个特定的设备具有设备相关的特性,就可以在多媒体设置程序中插入一个笔记本页。当多媒体设置程序创建了一个笔记本窗口时,它要检查 MCI_SYSINFO_LOGDEVICE 数据结构的 ulDeviceFlag 字段中的 MCI_SYSINFO_DEVICESETTINGS 风格位(style bit)。这个数据结构如图 2-44 所示。其中包含了有关安装在系统中的逻辑设备的信息。MCI_SYSINFO_DEVICESETTINGS 风格位表示 MCD 拥有用户自定义的设备设置页。

```
typedef struct_MCI_SYSINFO_LOGDEVICE{
  CHAR    szInstallName[MAX_DEVICE_NAME];      /* Device install name              */
  USHORT  usDeviceType;                        /* Device type number               */
  ULONG   ulDeviceFlag;                        /* Flag indicating whether device   */
                                               /* device is controllable or not    */
  CHAR    szVersionNumber[MAX_VERSION_NUMBER];/* INI file version number          */
  CHAR    szProductInfo[MAX_PRODINFO];         /* Textual product description      */
  CHAR    szMCDDriver[MAX_DEVICE NAME];        /* MCI driver DLL name              */
  CHAR    szVSDDriver[MAX_DEVICE_NAME];        /* VSD DLL name                     */
  CHAR    szPDDName[MAX_PDD_NAME];             /* Device PDD name                  */
  CHAR    szMCDTable[MAX_DEVICE_NAME];         /* Device-type command table        */
  CHAR    szVSDTable[MAX_DEVICE_NAME];         /* Device-specific command table    */
  USHORT  usShareType;                         /* Device sharing mode              */
  CHAR    szResourceName[MAX_DEVICE NAME];     /* Resource name                    */
  USHORT  usResourceUnits;                     /* Total resource units available   */
                                               /* for this device                  */
  USHORT  usResourceClasses;                   /* Number of resource classes for   */
                                               /* this device                      */
  USHORT  ausClassArray[MAX_CLASSES];          /* Maximum number of resource       */
                                               /* units for each class             */
  USHORT
ausValidClassArray[MAX_CLASSES][MAX_CLASSES];/* Valid class combination */
} MCI_SYSINFO_LOGDEVICE;
```

图 2-44 MCI_SYSINFO_LOGDEVICE 数据结构

如果 MCD **创建了一个或多个用户自定义的设置页,则多媒体设置程序就向** MCD **发一条** MCI_DEVICESETTINGS **消息。这就给予了** MCD **向某特定设备的多媒体设置笔记本插入设备相关的页的机会。若存在一个制造商提供的驱动程序**(VSD),**则** MCD **就把该消息传给它。一些设备特性与设备类型有关,还有一些设备特性则与制造商提供的具体硬件有关。**

MCI_DEVICESETTINGS_PARMS **数据结构**(**如图** 2-45 **所示**)**定义了要随** MCI_DEVICESETTINGS **消息传给** MCD **的信息。**hwndNotebook **字段包含了一个** CUA* **笔记本控制窗口的窗口句柄,页就将被插入在此处。**usDeviceType **字段定义了媒体设备的类型。**pszDeviceName **则定义了一个设备的逻辑设备名**(WAVEAUDIO01),**这个设备就是所要插入的。**

```
typedef struct_MCI_DEVICESETTINGS_PARMS{
ULONG     hwndCallback;          /* Window handle             */
HWND      hwndNotebook;          /* Handle to notebook window */
USHORT    usDeviceType;          /* Device type               */
PSZ       pszDeviceName;         /* Device name               */
 } MCI_DEVICESETTINGS_PARMS;
```

图 2-45 MCI_DEVICESETTINGS_PARMS 数据结构

你还必须为笔记本中每插入的一页提供帮助。推荐采用以下办法:在对话框过程中显式地处理 WM_HELP 消息,然后向由这一(些)页所创建的帮助实例发一条 HM_DISPLAY_HELP 消息。

当要求关于 Tab 的帮助时,多媒体设置程序向该页发一条 MM_TABHELP 消息(如图 2-46 所示)。页窗口过程可以显示相应的帮助面板。

```
MM_TABHELP
    mp1          ULONG ulPageID       /* Page identifier          */
    mp2          ULONG Reserved
    return       TRUE for handled and FALSE not handled
```

图 2-46　要求关于 Tab 的帮助

一些页使用 MCD 的设备相关参数来保存设置信息。设置新页的时候应当考虑到可能会有其它的页也使用这些设备相关的参数来保存它们的设置,因而也就会有或将会有其它的关键字存在。新页必须应当保存它所不认识的关键字的值。

图 2-47 给出的源代码创建了一个无模式的二级窗口,并在笔记本中插入一页。这段代码可以以外部 DLL 的一部分的形式给出,或者直接就是 MCD 代码自身的一部分。新页必须与已有笔记本风格一致,并且按照笔记本中的标准页的样子插入。

```
HWND InsertExamplePage(LPMCI_DEVICESETTINGS_PARMS pMCIDevSettings)
{
  HWND   hwndPage;                /* Page window handle          */
  CHAR   szTabText[CCHMAXPATH];   /* Buffer for tab string       */
  ULONG  ulPageId;                /* Page Identifier             */
 /******************************************************************/
 /*  Load a modeless secondary window.                             */
 /******************************************************************/
  hwndPage   = WinLoadSecondaryWindow(
                  pMCIDevSettings->hwndNotebook,
                  pMCIDevSettings->hwndNotebook,
                  ExamplePageDlgProc,
                  vhmodMRI,
                  ID_EXAMPLE,
                  (PVOID)pMCIDevSettings);

    if (!hwndPage) return(NULL);
  ulPageId = (ULONG)WinSendMsg(pMCIDevSettings->hwndNotebook,
                  BKM_INSERTPAGE,
                  (ULONG)NULL,
                  MPFROM2SHORT(BKA_AUTOPAGESIZE|BKA_MINOR,BKA_LAST));
```

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ *  Associate a secondary window with a notebook page .           */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
WinSendMsg (pMCIDevSettings - > hwndNotebook, BKM_SETPAGEWINDOWHWND,
          MPFROMP(ulPageId),MPFROMLONG(hwndPage));
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ *  Get tab text from DLL .                                       */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
WinLoadString(WinQueryAnchorBlock(HWND_DESKTOP),vhmodMRI,
      (USHORT)IDB_EXAMPLE,CCHMAXPATH,szTabText);
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ *  Set tab text .                                                */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
WinSendMsg(pMCIDevSettings - > hwndNotebook, BKM_SETTABTEXT,
      MPFROMP(ulPageId),szTabText);
return(hwndPage);
}
typedef struct{
   HWND hwndHelpInstance;
} MMPAGEINFO;
typedef MMPAGEINFO * PMMPAGEINFO;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ * Modeless secondary window procedure                            */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
MRESULT EXPENTRY ExamplePageDlgProc(HWND hwnd,USHORT msg,MPARAM,mp1,MPARAM mp2)
   PMMPAGEINFO pMMPageInfo = (PMMPAGEINFO) WinQueryWindowPtr(hwnd,QWL_USER);
   switch(msg){
     case WM_INITDLG:
       / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
       / *  Place window initialization code here .                       */
       / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
       pMMPageInfo = (PMMPAGEINFO)malloc(sizeof(MMPAGEINFO));
       WinSetWindowPtr(hwnd,QWL_USER,pMMPageInfo);
       / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
       / *  Create a help instance .                                      */
       / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
       pMMPageInfo - > hwndHelpInstance = WinCreateHelpInstance( ...);
       break;
```

```
case WM_DESTROY:
   /********************************/
   /* Clean up page window resources.                           */
   /********************************/
   WinDestroyHelpInstance(pMMPageInfo->hwndHelpInstance);
   free(pMMPageInfo);
   break;
case WM_COMMAND:
   /********************************/
   /* Process all commands.                                     */
   /********************************/
   return ((MRESULT)FALSE);
   break;

case MM_TABHELP:
   /********************************/
   /* Display help for a tab.                                   */
   /********************************/
   if (pMMPageInfo->hwndHelpInstance) {
      WinSendMsg(
            pMMPageInfo->hwndHelplnstance,
            HM_DISPLAY_HELP,
            MPFROMSHORT(WinQueryWindowUShort( hwnd,QWS_ID) ),
            HM_RESOURCEID);
   }
   break;

case WM_CLOSE:
   return ((MRESULT)FALSE);
   break;

case WM_HELP:
   if (pMMPageInfo->hwndHelpInstance) {
      WinSendMsg(
            pMMPageInfo->hwndHelpInstance,
            HM_DISPLAY_HELP,
            (MPARAM)mp1,
            HM_RESOURCEID);
   }
   return((MRESULT)TRUE);
   break;
case WM_TRANSLATEACCEL:
   return(WinDefWindowProc(hwnd,msg,mp1,mp2));
   break;
```

```
    case HM_QUERY_KEYS_HELP:
        return((MRESULT)IDH_HELPFORKEYS);
        break;
}
return (WinDefSecondaryWindowProc(hwnd,msg,mp1,mp2));}
```

图 2-47　**插入一个笔记本页**

第 3 章　流 处 理 器

本章介绍了 MCD 用来实现数据流动和同步的流编程接口(Stream Programming Interface, 简写为 SPI) 以及流处理器用于数据传输的接口。

3 .1　流处理器的体系结构

在同步/ 流管理器(ssm)对数据缓冲区和同步数据进行协调和集中管理的条件下, 成对的流将数据从源设备传输到目标设备。

流处理器可以被构造成第 0 环上运行的设备驱动程序, 或第 3 环上运行的动态连接库(DLL)。参阅 3 .7 节的“ DLL 模型: 文件系统流处理器 ”和 3 .8 节的“ 设备驱动程序模型:视频 PDD ”中的这两种类型。

有的流是由流处理器和物理设备驱动程序之间的直接连接而完美控制的, 而其它的流则未与映射特定物理设备的源数据或目标数据相联系。例如, 文件系统流处理器是一个 DLL, 因为所有的文件系统 I/ O 功能都能由服务于所有文件系统设备的第 3 环 OS/ 2 函数来实现。

当你决定了所需的流处理器类型后(DLL 或 DD), 你必须把所需的功能模块加入你的流处理器设计中。参阅图 3-1, 所示的是 DLL 和设备驱动程序流处理器的基本模块。

图 3-1　流处理器结构

尽管构造 OS/ 2 DLL 和设备驱动程序的详细编程过程很不相同,这并不影响流处理器的大部分逻辑结构。除极少数情况外,DLL 流处理器和设备驱动程序流处理器的结构是极其相似的。

图 3-1 描述了同步/ 流管理器(SSM)的逻辑结构以及它与流处理器的关系。同步/ 流管理器 DLL 向更高级的多媒体器件(例如媒体驱动程序)提供 SPI 服务,并提供流管理帮助器(stream Manger Helper,简写为 SMH)消息以支持流处理器 DLL。附加的 SMH 消息由 SSM 设备驱动程序使用标准 OS/ 2 内部设备驱动程序通信(IDC)接口提供给流处理器设备驱动程序(IDC 接口由函数 DevHelp. AttachDD 创建)。

表 3-1 介绍了 OS/ 2 多媒体提供的流处理器。

注意:关于对 OS/ 2 多媒体提供的流处理器的高级设计和使用,参阅附录 A“ 流处理器模块定义”。

表 3-1　OS/ 2 多媒体提供的流处理程序

流处理器	描　　述
音频	与售主特定驱动程序的 VSD 接口。支持 PCM,MIDI,ADPCM 格式。
MIDI 影射文件系统	用选定的 MIDI 影射过滤数据。利用文件系统服务从相关设备读/ 写数据。
多道	读取和分离交错数据。
视频	与 CODEC 一起输出视频数据到显示器。
系统内存	从内存缓冲区存/ 取数据。参见 3.4 节“ 尾接提示点事件支持”。
CD-ROM XA	读取 CD-ROM XA 数据并从视频和数据区中分离出音频区。
CD-DA	直接从 CD-ROM 驱动程序读取数字音频数据。可用于通过声卡重放 CD 音频数据。

3.2　同步的特点

对多媒体应用来说,具有使事件同步的能力是极其重要的。例如,在播放一个音频波形(“ 星条旗”中那种特定的钹撞击声)的同时,你可能想显示一幅特定的位图(一幅燃放烟火的图象)。从标准的 OS/ 2 应用角度看,两个独立的线程或单独一个线程可能能控制这些事件。但是,它们都不能保证两个事件在一个特定的时间间隔内同时发生。两个事件之间的延迟时间越长,用户就会越明显地感觉到不同步。

通过确保像输出指定数据元素(数字音频、图象、MIDI 音频等)之类的程序事件能在极短的实时时间限制内实现同步的方法,同步/ 流子系统的设计支持多媒体应用程序中所需的同步。同步/ 流子系统同时还降低了事件同步所需的代码的复杂性。

对多媒体应用程序开发人员来说,使用户指定的事件和系统驱动的事件同步也是很重要的。例如,在响应用户移动鼠标或按键盘上的光标键(↑↓)时,应用程序可能需要调整音乐重放音序的步调。此时,任何可觉察到的响应延迟都是不可接受的。更重要的情形是,在一个飞行员训练程序中,受训者必须按下一个键来响应接近地面的警报。如果受训者感觉到了在按键和警报声变化之间有延迟,那么可以合理地认为实际的座舱控制也是如此。错误的预测也许就会导致现实中的坠机事件。

SSM 的同步功能简化了程序员的工作。例如,程序员不用去创建庞大而复杂的控制结构来同步一组流,而只需把这些流看作是组中的部分。

同步/ 流子系统的设计有以下几个特点,它们提供了对实时事件的简单有效的同步控制:

· 主/ 从关系;

· 同步脉冲的产生;

· 同步脉冲的处理;

· 同步/ 流子系统事件;

· 空的流处理器。

以上这些特点利用了 OS/ 2 的多任务处理能力,从而避免了将物理内存中特定的代码和数据区转移到硬盘上,这确保了所需模块能以实时模式处理数据和请求,消除了交换所需关键区可能引起的延迟稳患。

3.2.1 主/ 从关系

主/ 从关系是指控制同步化事件的一系列命令的清单。对应关系是 1∶N,即一个目标(同步主体)控制一个或多个附属目标(从体)的行为。这种关系要用 SpiEnableSync 函数建立,其中指定一个数据流为主体,指定一个或多个数据流为从体。同步/ 流管理器(SSM)把实时信息从主体传送到所有的从体,每个从体从中可以获得基于 MMTIME 标准(1/ 3 毫秒)的当前时间。时间信息(即同步脉冲)使每个从体流处理器调节流的活动以便达到同步。

主/ 从同步关系遵循以下规则:

· 一个数据流只能在一个同步关系中充当从体。

· 流处理器必须能控制属于不同同步主体的多个数据流(如果支持多数据流的话)。例如,在处理器控制下的每个活动流可以有不同的主体。

· 一个流不能同时是两组独立从体的主体(即任何附加到已存在的主/ 从组的流只能作为从流)。

· 通过使用以下 SPI 函数的“slave”标志,同步组可以作为一个整体来启动、停止和查找:

SpiStartStream,SpiStopStream,SpiSeekStream。

· 同步关系中的任何从体可以被单独启动或停止,而不影响主体和其它从体的活动。可以在不顾组中从体的情况下单独停止主体,也可选择在停止主体的同时停止所有的从体。

· 同步化组中的流时间总是与主体的流时间保持一致,包括被停止和重新启动的从体流。

· 流的查找是基于单个流来说的。比如,主体流中的查找不会跑到同组的其它从体流中去。

一个从体也可能会保持不了同步。这种情况称为同步过速(syncoverrun)。当流处理器还未处理完从 SSM 来的最后一个同步脉冲而又收到下一个脉冲的时候,就会发生同步

过速。应用程序可以选择要求被通知任何同步过速的发生(通过 SpiEnableEvent 函数)。一旦应用程序接收到同步过速事件,流此时并未停止但应用程序可能要求停止。

3.2.2 同步脉冲的产生

同步脉冲代表的是当前流同步主体的时钟值及其流处理器(HSTREAM)。时钟值的单位是 MMTIME(1/3 毫秒)。当它为 0 时代表的是从启动与主体相联系的多媒体数据目标开始直到主体流被启动时的时间。如果正在对主体流进行查找操作,该流必须被停止。于是查找时主体流时间也被停止,并复位到查找点的流时间。当流被重新启动后,流时间从查找点重新开始。这意味着流时间对应的是数据的位置而不是流处于活动状态的时间长短。

同步脉冲由主体按常规产生。只有当同步/流管理器(SSM)认为从体与主体不同步时,从体才能接收到同步脉冲。流管理器按照程序确定的流同步关系发送同步脉冲。对 DLL 和设备驱动程序的从体流,这种发送方法都是有效的。

每个从体流处理程序必须定期用它认为的流时间更新同步脉冲 SYNC_EVCB。同步/流管理器(SSM)通过核对从体处理器流时间和主体流时间来确定是否应给从体处理器发送同步脉冲。

设备驱动程序流处理器通过它们的同步脉冲事件管理块(SYNC_EVCB)来接收同步脉冲。通过轮询同步脉冲 SYNC_EVCB 中的一个标志,设备驱动程序流处理器必须查对 SSM 来的同步脉冲。SSM 设置该标志以表示要有一个同步脉冲,并更新当前的主体流时间。通常,在中断处理过程中,设备驱动程序从体处理器轮询该标志一次,并相应调整流。

DLL 流处理器可通过两种方法接收同步脉冲。一种方法是与 SSM 之间建立一个信号,另一种方法同设备驱动器流处理器所用的方法。

3.2.3 同步脉冲处理

每个流处理器(DLL 或设备驱动程序)都能处理同步脉冲以支持活动流的同步。典型的情况是,流处理器可以根据同步脉冲 SYNC_EVCB 所接收的信息,按实时情况校正活动流的进展。其中的 SYNC_EVCB 由流处理器创建,并由 SSM 来进行访问。

例如,如果从体的当前本地时间是 45000(以 MMTIME 为单位,即 15.0 秒),而主体的同步脉冲时间是 44500(14.83 秒),于是从体流处理器的同步脉冲逻辑应调整数据流动的步调(减慢速度,或重复最后 170 毫秒的数据操作)。这种时间调节(resync)逻辑只适用于处理器中正在流动的数据类型。

由于系统有时任务繁多,使 DDL 同步脉冲逻辑在两个同步脉冲之间不能取得控制(任务调度的延缓偶尔还会延缓实时线程),则处理器总是取最当前的主体时间(由最后一次同步脉冲传送)。如果在下个同步脉冲到来之前还未处理当前脉冲,SSM 设置同步脉冲的过速标志(overrun flag)。同时,同步脉冲逻辑以及所有相关的控制数据目标必须保存在固定的内存中,以避免这些代码和数据页被页出(至少在流处于活动状态时不会)。

同步脉冲逻辑从根本上说必须是高性能的。复杂、耗时而又带着长路径的同步调整例程如果被连续调用的话,将不能保持同步。同步脉冲逻辑是次级中断处理器,因此必须对它的工作进行优化。当流不需要进行调整时,就根本不需调用调整例程,因为调整和不调整的结果是一样的。

同步/ 流管理器(SSM)具有同步脉冲发送逻辑,用来控制什么时候该设置从体设备驱动程序处理器的同步脉冲 SYNCPOLLING 标志位(SSM 不调用处理器的 IDC 接口)。该逻辑同样还控制着什么时候调度 DLL 处理器的实时线程(即清除一个线程正服务的信号)。典型情况是,通过轮询同步脉冲 SYNC_EVCB 中的 SYNCPOLLING 标志位,DLL 处理程序可以检测到同步脉冲的产生。其中 SYNC_EVCB 是处理程序在回应 SHC_ENABLE_SYNC 消息时传送给 SSM 的。只有当主体流时间和从体的本地时间(均存于同步脉冲 SYNC_EVCB 中)相差超过指定的同步容错值(resync tolerance value)时,才设置 SYNCPOLLING 位。当从主体接收到同步脉冲后,SSM 为所有的从体确定时间差,并且只通告那些其流偏离主体流时间过多的从体。每个能接收同步脉冲的流处理器都必须提供各自的容错值(一个以 MMTIME 表示的最小时间间隔),该值存于流的 SPCB 中并作为从 SHC_CREATE 返回时的传送值。

3.2.4 同步/ 流子系统事件

同步/ 流子系统事件是指主体或从体流处理器状态的特定改变。其中一些事件对同步并无任何影响,而只是表明数据流的状态。

定义了两类事件:确定事件和绝对事件。绝对事件是所有流处理器都必须支持的事件(例如流结束或完成了预卷动)。确定事件是只有一部分处理器支持的事件(例如对于某种特定数据类型的常规事件)。应用程序自动获得绝对事件的通知;但是,如果应用程序想接收其它事件通知,必须用 SpiEnableEvent 来允许确定事件。事件将保持允许直至被禁止允许。只有 SSM 和流处理器能产生事件。

表 3-2 列出了同步/ 流子系统当前所定义的事件。流处理器也可以定义新事件,作为流处理器和应用程序之间的通信手段。参阅 EVCB 数据结构中的 ulType 和 ulSubType 字段,见《OS/ 2 多媒体编程指南》一书。

表 3-2 同步/ 流子系统事件

事　　件	数据结构	描　　述
EVENT_CUE_DATA(显式)	DATA_EVCB	对流中一些特定数据区的尾接提示点。可允许该事件为单个事件或重复事件。
EVENT_CUE_TIME (显式)	TIME_EVCB	对流启动后的流时间的尾接提示点。可允许该事件为单个或重复事件。重复事件是以时间间隔定义的事件,每个时间间隔内产生一次该事件。单个事件将保持允许状态,即使它们被报告。若在尾接提示点以前流已被及时回找,操作继续,则尾接提示点将被再次报告。

续表

事　　件	数据结构	描　　述
EVENT. CUE. TIME. PAUSE (显式)	TIME. EVCB	对流启动后的流时间的尾接提示点。当到达该尾接提示点时,流将被暂停。该事件可被允许为单个事件。
EVENT. DATAOVERRUN (显式)	EVCB	流处理器检测到了数据过速。在此时记录的数据可能会丢失。
EVENT. DATAUNDERRUN (隐式)	EVCB	目标流处理器检测到了欠载的情况,即没有数据可输出到输出设备。通常在这种情况下,目标流处理器将试图得到另一个缓冲区,然后暂停设备。当更多数据可被输出时,目标流处理器将被重新启动。这种情况造成了输出数据流的一次中断。使用交错数据格式时,如果已到数据尾但还没到文件尾,则会引起欠载。
EVENT. EOS(隐式)	IMPL. EVCB	流事件结尾。当目标流处理器用完了流中的最后一个缓冲区后,就会产生该事件。这告诉应用程序流处理已经结束了。
EVENT. ERROR (隐式)	IMPL. EVCB	流动过程中产生了一个错误。
EVENT. PLAYLISTCUE-POINT(隐式)	PLAYL. EVCB	存储器流处理器演播表的尾接提示点事件。
EVENT. PLAYLISTMESSAGE (隐式)	PLAYL. EVCB	存储器流处理器演播表消息。
EVENT. QUEUE. OVERFLOW (隐式)	IMPL. EVCB	事件队列溢出。这表明由于产生了太多事件,一个事件已经丢失。应用程序(MCD)应当用该事件来清除所有等待状态。
EVENT. STREAM. STOPPED (隐式)	IMPL. EVCB	流已被置空或废弃。(参见《编程指南》中的 SpiStopStream 函数)。
EVENT. SYNC. PREROLLED (隐式)	IMPL. EVCB	所有的同步化流都被预卷动。(参见《编程指南》中的 SpiStartStream 函数)。
EVENT. SYNCOVERRUN (显式)	OVRU. EVCB	检测到了流中的同步过速。

SSM 通过媒体驱动程序把事件传送给应用程序。这通过对一个事件例程进行回调,该例程是应用程序通过 SpiCreatStream 调用在它和 SSM 之间注册的。把该事件例程看作一个次级中断例程来使用,不要企图进行大量的操作。该事件例程是单过程处理的,它只能同时接收一个过程。因此,事件必须一个一个地处理。

由于绝对事件只有一个 EVCB,因此较好的方法是,把所需的信息从 EVCB 拷贝到局部变量中等待事件例程处理。事件例程有以下接口:

```
Event_Entry(PEVCB,pEVCB,rc)
```

3.2.5 空的流处理器

SSM 提供了一种创建空流的能力,使多媒体应用程序可以让非流设备与流设备同步。空流是为非流设备创建和启动的,但它们之间没有数据流的联系。二重唱演奏器 II 样例程序中提供了一个同步非流的 CD-DA 设备的实例。

任何应用程序或媒体驱动程序都能创建一个或多个空流。由于空流没有预置的同步容错值,创建模块必须通过先后调用 SpiGetProtocol 函数和 SpiInstallProtocol 函数来设置该值,也可在两次调用之间修改同步容错字段。这样也就设置了对调用过程通知事件的频率,使系统既能保持同步,又不必在每个同步脉冲时运行通知线程。

一个给定的过程可以创建多个空流。和通知实流处理器的方法一样,这些空流也可在同步脉冲时被通知。因此,每个空流处理器必须支持一些实流处理器所支持的功能,如此产生的每个空流可以具有不同的同步脉冲定时特性。通过改变各自 SPCB 中的同步容错值,可以控制每个流的同步脉冲粒度。

空流处理器必须支持以下的流处理器命令(SHC)消息:

- SHC_CREATE
- SHC_DESTROY
- SHC_DISABLE_EVENT
- SHC_DISABLE_SYNC
- SHC_ENABLE_EVENT
- SHC_ENABLE_SYNC
- SHC_ENUMERATE_PROTOCOLS
- SHC_GET_PROTOCOL
- SHC_GET_TIME
- SHC_INSTALL_PROTOCOL
- SHC_NEGOTIATE_PROTOCOL
- SHC_SEEK
- SHC_START
- SHC_STOP

3.3 流 协 议

流协议控制快(SPCB)定义了控制数据流活动的关键性参数。应用程序可以查询、安装或取消一个流处理器中指定的 SPCB。SPCB 信息以资源文件的方式安装在 SPI.INI 文件中(关于如何安装流协议,可参阅 5.6 节的"安装流处理器")。

每个流处理器支持一个或多个流协议。每个 SPCB 是通过流数据类型值(SPCB 中的关键性参数)来唯一区分的。SPCB 关键参数中的一个字段允许流处理器拥有多个安装了同一数据类型的 SPCB。应用程序可以使用该字段指定它想使用的 SPCB,不管是何种

数据类型。系统中的每个应用程序可以为同一数据类型定义多个 SPCB(参阅 SPCBKEY 数据结构中的 ulType 字段)。应用程序可以通过取消老的 SPCB 以及安装新的 SPCB 来修改流协议。

SPCB 中的参数:

字　段	含　义
SPCB Length	SPCB 结构的长度,用于将来扩充 SPCB。
SPCB Key	数据流类型和内部关键字段。内部关键字段用于区分多个同一类型数据流的 SPCB。 · Data Type——流的数据类型(例如,波形或 MIDI); · Data Subtype——流的子类型(例如 16 位立体声 PCM)。 · Internal Key——用于区分多个同一数据类型和子类型的 SPCB。
Data Flag	数据类型属性。(即指明该数据类型是否支持数据或时间的尾接提示点。) · SPCBDATA. CUETIME——该数据类型支持时间尾接提示点事件。 · SPCBDATA. CUEDATA——该数据类型支持数据尾接提示点事件。 · SPCBDATA. NOSEEK——不能对该数据类型进行查找。
Number of Records	每个缓冲区中的最大记录数(仅对分离流有效)。
Data Block Size	块的大小。块是数据的原子单位。例如,对于采样速率为 44 .1kHz 的 16 位立体声数字音频数据类型 PCM,块大小应为 4 个字节。)
Data Buffer Size	流处于流动状态时使用的缓冲区大小,最大值为 64KB。
Min Number of Buffers	维持一个恒定的数据流所需缓冲区的最小数目。
Max Number of Buffers	所需缓冲区的最大数目(理想数)。对于标准流,这是指分配给流的缓冲区数。而对用户提供的缓冲区流动,这是指 SSM 能为使用者争取到的缓冲区数。由于源流处理器要反复向 SSM 传送同一缓冲区集合,它也可利用该值。如果把缓冲区数设置为集合中的缓冲区数减 1,源流处理器就可以得知在什么时候目标已用完了一个缓冲区,即该缓冲区可被再利用了。假设缓冲区集是一个有序集,且每次都是以同样的顺序使用每个缓冲区。
Source Start Number	启动源流处理器所需的空缓冲区数。该值必须至少等于源流处理器可能要求的最大缓冲区数。
Target Start Number	启动目标流处理器所需的满缓冲区数。该值必须至少等于目标流处理器可能要求的最大缓冲区数。通常一个目标在启动时至少需要两个缓冲区。
Buffer Flag	缓冲区属性(即用户提供缓冲区、固定块大小、交错数据类型以及最大缓冲区大小)。 · SPCBBUF. USERPROVIDED——用户为流动提供的缓冲区。SSM 不会分配缓冲区,但将试图锁定用户提供的缓冲区或把数据拷贝到锁定的缓冲区中。使用该标志将会影响流的工作。只有源流处理器可以设置该标志。该标志与 SPCBBUF. INTERLEAVED 标志互斥。设置了该标志后,不能使用 SPCBBUF. FIXEDBUF。

- SPCBBUF. FIXEDBUF——流处理器的缓冲区(对该数据类型来说)是大小固定的。该标志不能与 SPCBBUF. USERPROVIDED 一起使用。SPCBBUF. INTERLEAVED 标志(分离流)隐含设置了 SPCBBUF. FIXEDBUF。
- SPCBBUF. NONCONTIGUOUS——每个数据缓冲区分配在相邻的物理内存,除非两个流都设置了不相邻标志(SPCBBUF. NONCONTIGUOUS)。该标志使系统可以灵活地分配内存。一个设备驱动程序流处理器可能需要相邻的内存,而 DLL 流处理器可能不用。
- SPCBBUF. INTERLEAVED——该流是一个分离流。它包含一个含交错数据的输入流,该流被分离成了几个单独数据类型的流。只有源流处理器可以设置该标志。该标志与 SPCBBUF. USERPROVIDED 标志互斥。设置了该标志后,不能使用 SPCBBUF. FIXEDBUF。
- SPCBBUF. MAXSIZE——ulBufSize 字段代表了该流处理器所能处理的最大缓冲区。
- SPCBBUF. 16MEG——可分配超过 16MB 的流缓冲区。能够支持超过 16MB 地址的流处理器设备驱动程序使用该标志。

Handler Flag　　流处理器标志(即处理器能产生和接收同步脉冲,把 SSM 计时器作为主体使用,非流处理器)。

- SPCBHAND. GENSYNC——该流处理器能产生同步脉冲。
- SPCBHAND. RCVSYNC——该流处理器能接收同步脉冲。
- SPCBHAND. NONSTREAM——该流处理器是一个非流处理器(也就是说,是一个可以参与同步但不能流动的流处理器)。
- SPCBHAND. GENTIME——该流处理器能与实流时间保持一致。该处理器也支持 SpiGetTime 和尾接提示点事件。
- SPCBHAND. NOPREROLL——该流处理器不能预卷动其设备(也就是说,为了记录流,源流处理器不能被预卷动)。如果要求预卷动该流时,SSM 会把该流看作已经预卷动过了)。
- SPCBHAND. NOSYNC——该流处理器可以在一个同步组中但不接收或产生同步脉冲。(有利于把流合成一组,以便作为一个整体来进行控制。)
- SPCBHAND. PHYS. SEEK——该流处理器对物理设备进行查找。其它的流处理器在一个查找请求时只是调整一下流时间。当调用 SpiSeekStream 函数时,SSM 总是首先调用该流处理器。

Resync Tolerance　　同步容错值。对于从体,该值用于判断是否应给从体流处理器发送一个同步脉冲。

Sync Pulse Granularity　　两个同步脉冲之间的时间间隔,以 MMTIME 为单位。如果该流处理器是主体,则该值用于保存同步脉冲的产生粒度,但不能产生自己的同步脉冲。

Bytes Per Unit　　单位时间字节数。用于查找非压缩或变长度的线性数据。也可用于流处理器中的 SHC. GETTIME 查询。

MMTIME Per Unit　　每单位代表的 MMTIME 数。同样也用于查找和流处理器的"get time"函数。

关于结构和标志定义的进一步信息,参阅《OS/2 多媒体编程指南》一书。

3.3.1 创建流时的流协议协商

以下是关于流协议协商的一些描述:

· 协商后的 SPCB 中的以下标志将被设置,流处理器应按标志来工作。源或目标的协商后的 SPCB 将包含以下位标志:

SPCBHAND_GENSYNC SPCBHAND_RCVSYNC

SPCBHAND_GENTIME SPCBHAND_PHYS_SEEK

· 在协商过程中 DATATYPE_GENERIC 将与其它任何数据类型匹配。这对于像文件系统处理器之类的流处理器很有用,因为它们几乎可以是任何数据类型的源。

· 如果 SPCB 的主要数据类型都不是通用的,则数据类型和子类型字段必须相等,否则会发生错误。内部关键字段在协商期间不可用。协商将返回内部关键字段为 0。

· 如果未指定,则块大小将隐含设置为 1 个字节。源和目标块大小必须相等,否则协商失败。

· 数据缓冲区大小必须是块大小的倍数。

· 如果一个流处理器的固定缓冲区大小(SPCBBUF_FIXEDBUF)比另一个处理器的最大缓冲区大小(SPCBBUF_MAXSIZE)还要大,则协商失败。

· 两个处理器不能有不同长度的固定缓冲区大小(SPCBBUF_FIXEDBUF)。

· 两个处理器不能有不同长度的最大缓冲区大小(SPCBBUF_MAXSIZE)。

· 协商将隐含设置固定缓冲区大小(SPCBBUF_FIXEDBUF)。否则,缓冲区大小将设置为两个 SPCB 缓冲区大小中较大的一个,但不超过最大缓冲区大小(SPCBBUF_MAXSIZE),如果该大小被指定了的话。

· 如果没有指定特殊情形(SPCBBUF_FIXEDBUF, SPCBBUF_MAXSIZE, SPCBBUF_USERPROVIDED),最大缓冲区大小将用于创建流。

· 对用户提供的缓冲区(SPCBBUF_USERPROVIDED),缓冲区大小将设为最大缓冲区大小(SPCBBUF_MAXSIZE),或是圆整到块大小倍数的最大可能的缓冲区大小。

· 分离流(SPCBBUF_INTERLEAVED)必须有一个每个缓冲区的最大记录数,该值必须大于 0。由于只有源流处理器能设置该值,因此目标流处理器必须把该字段设为 0。

· 最小和最大缓冲区数必须大于 0。

· 最小缓冲区数用可用的最大值。

· 最大缓冲区数用可用的最大值。

· SSM 将试图分配所要求的最大数目的流缓冲区。如果空间不够,但够分配所需的

最小数目的缓冲区,就创建该流。否则,由于资源不够,流的创建将失败。

· 启动源所需的空缓冲区数目总是从源 SPCB 中得到。
· 启动目标所需的满缓冲区数目总是从目标 SPCB 中得到。
· 对于 SpiGetTime 请求,每个处理器必须确定它能否接受请求并返回实时信息。通过指定 SPCBHAND_GENTIME 标志可做到这一点。对于协商,目标流处理器隐含的信息提供者,除非只有源能提供这一信息。
· 每单位的字节数和 MMTIME 数由流处理器设置,该程序在每条语句后处理实时请求。
· 每个处理器必须确定能否发送或接收同步脉冲。这可以通过指定 SPCBHAND_GENSYNC 或 SPCBHAND_RCVSYNC 标志来实现。对于协商,目标流处理器是隐含的同步脉冲发生者和接收者,除非只有源能产生和接收同步脉冲。
· 同步容错值只对设置了 SPCBHAND_RCVSYNC 标志的处理器有效。
· 同步脉冲粒度只对设置了 SPCBHAND_GENSYNC 标志的处理器有效。
· SPCBHAND_PHYS_SEEK 用于指定流处理器在一个 SHC_SEEK 调用时是否进行物理设备的查找。SSM 利用该信息决定在 SpiSeekStream 调用时应首先调用哪个流处理器。否则,指定了 SPCBHAND_GENTIME 标志的流处理器将被最后调用。
· 任何的保留字段必须设为 NULL。
· 流协议中所有位标志中的未定义位必须设为 0。

3.4 尾接提示点事件支持

在一些特定情况下,源流处理器向应用程序(或媒体驱动程序)报告一个尾接提示点事件是不合适的,而应由目标来报告尾接提示点。一般情况下,一个正在等待尾接提示点的应用程序希望在流目标正输出数据时接收到尾接提示点。系统内存流处理器(System Memory Stream Handler)即属于这种情况。

系统内存流处理器能够从演播表中流动数据。演播表基本上是一个小程序,它指定了用户应用程序中的哪些内存目标应该流动以及流动的顺序。其中包含的控制信息能够在演播表中实现循环,分支以及子程序调用。演播表的另一个特点是 CUEPOINT 指令,它使演播表可以指定在数据流中的何处应该给应用程序传送尾接提示点。由于演播表是在源流处理器内"播放"的,它必须通告目标流处理器在数据流的合适时间报告尾接提示点。

SSM 为源流处理器提供了一种能力,使它可以通过 SSM 把一个尾接提示点与它传送给目标流处理器的一个缓冲区联系起来。在目标流处理器请求之前,该缓冲区及其属性保存在数据流中。缓冲区的属性和缓冲区一起传送给目标流处理器(参阅《OS/2 多媒体编程指南》中的 SMH_NOTIFY 消息)。当目标流处理器接收到一个非 0 属性的缓冲区时,它必须对该事件作标记(例如 EVENT_CUE_DATA)。在向 SSM 进行 SMH_REPORTEVENT 调用时,属性值作为传递给事件的一个参数。

SSM 的这个特点支持内存演播表,即在尽可能在接近尾接提示点事件发生的实际时刻传送它。为此,必须由目标流处理器把尾接提示事件标记给应用程序,即媒体驱动程序。如果系统内存流处理器是源流处理器,它必须通告目标流处理器去标记一个尾接提示点事件。关于演播表的更多信息,参阅《OS/ 2 多媒体编程指南》一书。

3.5 CD-ROM XA 流处理器

SSM 提供了一种称为分离流的技术,即允许一个数据流源和多个目标。例如,CD-ROM XA 能在一个数据缓冲区内交错音频、视频和文本数据。CD-ROM XA 流处理器为 CD-ROM XA 设备把数据分离成多个流。更高效的方法是,流处理器先把混合数据读入一个缓冲区中,然后把视频数据插入到视频流,把波形音频数据插入到波形音频流中,而不必进行数据拷贝。于是波形流处理器就使用波形音频数据,而视频流处理器则使用视频数据。这样我们可以把波形音频流处理器仅看作是数据流的音频部分,而视频流处理器为视频部分。

一个数据流可被分离为多个分离流。分离流数只受系统资源的限制。流的每个独立的分离流是各自独立的。分析和判断不同数据类型的工作是由源流处理器完成的,它实际上是用源设备来的数据去填充 SSM 的一个空缓冲区。

源流处理器必须确定什么数据去什么流。然后它必须把缓冲区中各个记录的指针返回给 SSM。每个记录是缓冲区的一部分,其中包含了针对某一分离流的数据。这些记录返回给 SSM,由 SSM 来决定它们该去哪个流。然后 SSM 将在各个缓冲区中将记录排队或为分享该缓冲区的一个分离流记录队列。

你可以按下面方法创建分离流。首先必须创建多个流,每个流包含一个对 SSM 的 SpiCreateStream 函数调用。这些函数里的第一个是一个正规的流创建调用,其 hstream-Buf 参数设为 NULL。随后的其它 SpiCreateStream 调用需要与第一个流分享缓冲区,因此它们必须把第一个流的句柄赋给 hstreamBuf 参数。这就是 SSM 使用的技术,使这些分离流可以共享第一个流的缓冲区。

源流处理器在 SHC_CREATE 调用时也会接收到该参数,它需要支持这种流动。源流处理器通过在 SPCB 中设置的 SPCBBUF_INTERLEAVE0 来表明它是否能支持分离流。如果不能,它将拒绝让任何 SpiCreateStream 函数使用它的缓冲区。共享缓冲区的流将协商使用缓冲区,其大小和数目是与分配缓冲区的流一致。

分离流技术只适用于单源、多目标的情况。

3.6 流 动 方 案

以下是几个编程方案,提供了在 OS/ 2 多媒体环境下使用同步/ 流子系统函数进行数据流动的示范伪代码。这些伪代码描述了如何利用 SPI 函数去构造你的应用程序代码。关于同步/ 流子系统接口如何使用的详细介绍,可参阅《编程指南》中关于函数和消息的详细定义。

3.6.1 从文件系统流动波形音频数据

该方案描述了如何去创建一个波形音频数据流并将一个包含波形音频的文件与源流处理器(即文件系统处理器 DLL)联系在一起,此时的目标处理器是音频流处理器。如图 3.2 示。

```
/**********************************/
/*                MAIN            */
/**********************************/
 #include      <os2.h>
 #include      <stdlib.h>
 #include      <stdio.h>
 #include      <conio.h>
 #include      <string.h>
 #include      "os2me.h"


main()
{
RC             ulRC;           /* Error return code */
SPCBKEY        SPCBkey;        /* Data type to stream */
IMPL_EVCB      EVCB;           /* Event control block for implicit events */
ACB_MMIO       acb;            /* Associate control block used to assoc */
                               /* The file to stream (play) */
DCB_AUDIOSH    dcb;            /* Audio device driver information */
HAND           ahand[5];       /* Enumerate handler info */
ULONG          ulNumHand = 5;
HID            hidSource,hidTarget,hidunused;/* Handler ID */
HSTREAM        hstream;        /* Stream handle */
HEVENT         hevent;         /* Event handle for implicit events */
HMMIO          hmmioIn;        /* Handle to mmio file open */
/* Get list of all stream handlers in system */
if (rc = SpiEnumerateHandlers(&ahand, &ulNumHand))
   return(rc);  /* error ! */
/* Get the stream handler ID for the File System Stream Handler */
if (rc = SpiGetHandler("FSSH", &hidSource, &hidunused))
   return(rc);  /* error ! */
/* Get the stream handler ID for the Audio Stream Handler */
if (rc = SpiGetHandler("AUDIOSH$", &hidunused, &hidTarget))
   return(rc);    /* error ! */
/* Create a data stream from the file system to the */
/* waveform audio ACPA device connected to speakers.*/
SPCBkey.ulDataType = DATATYPE_WAVEFORM;
```

```
SPCBkey.ulDataSubType = WAVE_FORMAT_4S16;
SPCBkey.ulINtKey = 0;
strcpy(dcb.szDevName,"AUDIO1$");
dcb.ulDCBLen = sizeof(dcb);
dcb.hSysFileNum = (**hSysFileNum returned by audio dd on audio_init call)
if (rc = SpiCreateStream(hidSource,
                         hidTarget,
                         &SPCBkey,
                         NULL,
                         (PDCB)&dcb,
                         (PEVCB)&EVCB,
                         EventProc,
                         0,
                         &hstream,
                         &hevent))
   return(rc);  /* error! */
/* USE MMIO to access a waveform audio object */
if ((hmmioIn = mmioOpen("c:\lawton\files\waveform\hendrix.jim"
                        NULL,
                        MMIO_READWRITE|
                        MMIO_DENYNONE)) == NULL)
     return(ERROR_FILE_NOT_FOUND);
/* Fill in acb with data object info */
acb.ulACBLen = sizeof(ACB_MMIO);
acb.ulObjType = ACBTYPE_MMIO;
acb.hmmio = hmmioIn;
/* Associate the waveform data object with the source handler */
if (rc = SpiAssociate(hstream,hidSource,&acb))
   return(rc);    /* error! */
/* Start the streaming */
if (rc = SpiStartStream(hstream,SPI_START_STREAM))
   return(rc);    /* error! */
/* Do other processing */

/* Create a semaphore and block this thread on the semaphore */
/* until the streaming is complete                           */
                     .
                     .
                     .
/* Destroy the stream */
if (rc = SpiDestroyStream(hstream))
   return(rc);    /* error! */
```

```
/ * Perform any other resource cleanup */

/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ *                                Event Procedure                       */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
RC EventProc(EVCB * pEVCB)
{
/ * Handle events from the Sync/ Stream Manager */
Switch (pEVCB - > ulType){
  case EVENT_IMPLICIT_TYPE:
      switch(pEVCB - > ulSubType) {
         case EVENT_EOS:    / * End of Stream */
            / * Clear the semaphore so that the main */
            / * routine can continue execution .     */
             break;
         case EVENT_ERROR:  / * Error occurred */
            / * flag error condition in main line application */
             break;
         case EVENT_STREAM_STOPPED:  / * Discard/ Flush Stop */
            / * Do processing while stream stopped */
             break;
         case EVENT_SYNC_PREROLLED:    / * Stream is prerolled */
            / * Now that the producers are prerolled, do a real
                start to the stream handlers */
             break;
         default:
            / * Unknown event, do something */
         }
      break;
  default:
     / * Unknown event, do something */
  }
}
```

图 3-2　从文件系统流动波形音频

3.6.2　同步化的 MIDI 和波形流

该方案描述了如何创建两个流(波形流和 MIDI),并使它们互相同步。音频流是同步主流,波形流的源处理器是文件系统流处理器 DLL。MIDI 流的源处理器是文件系统流处理器 DLL。

在此情况下,音频流处理器不能作为一个主体流的从体,但可以作为一个主体流处理

器。如图 3-3 示。

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ *                                 MAIN                               */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
main()
{
RC             ulRC;              / * Error return code */

SPCBKEY        spcbkey;           / * Data type to stream */
IMPL_EVCB      evcb1,evcb2;       / * Event control block for implicit events */
ACB_MMIO       acb;               / * Associate control block used to assoc */
                                  / * The file to stream (play) */
DCB_AUDIOSH    dcb;               / * Audio device driver information */
HAND           ahand[5];          / * Enumerate handler info */
ULONG          ulNumHand = 5;
HID            hidSource1,hidTarget1,hidunused;    / * Handler IDs */
HID            hidSource2,hidTarget2
HSTREAM        hstream1,hstream2;/ * Stream handle */
HEVENT         hevent1,hevent2;  / * Event handle for implicit events */
HMMIO          hmmio1,hmmio2;     / * Handle to mmio file open */
SLAVE          slave[1];          / * Sync slave structure */

/ * Get list of all Stream handlers in system */
if (rc = SpiEnumerateHandlers( &ahand, &ulNumHand))
   return(rc);     / * error ! */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ * * * * * CREATE WAVEFORM AUDIO STREAM (Stream # 1)  * * * * * * * * * * * * */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ * Get the stream Handler ID for the file system stream handler */
if (rc = SpiGetHandler( "FSSH", &hidSource1, &hidunused))
   return(rc);     / * error ! */
/ * Get the stream handler ID for the Audio Stream Handler */
if (rc =  SpiGetHandler( "AUDIOSH $ ", &hidunused, &hidTarget1))
   return(rc);     / * error ! */
/ * Create a data stream from the file system to */
/ * the audio ACPA device connected to speakers .*/
spcbkey .ulDataType = DATATYPE_ADPCM_AVC;
spcbkey .ulDataSabType = ADPCM_AVC_HQ;
spcbkey .ulINtKey = 0;
strcpy(dcb .szDevName ."AUDIO1 $ ");
dcb .ulDCBLen = sizeof(dcb);
dcb .hSysFileNum = ( * * hSysFileNum returned by audio dd on audio_init call)
```

```
if (rc = SpiCreateStream(hidSource1,
                        hidTarget1,
                        &spcbkey,
                        NULL,
                        (PDCB)&dcb,
                        (PEVCB)&evcb1,
                        EventProc,
                        0,
                        &hstream1,
                        &hevent1))
   return(rc);    /* error! */
/* USE MMIO to access a waveform audio object */
if ((hmmio1 = mmioOpen("E:\Mygreat\waveform\file\screams.hq"
                       NULL,
                       MMIO_READWRITE|
                       MMIO_DENYNONE)) == NULL)
   return(ERROR_FILE_NOT_FOUND);
/* Fill in acb with data object info */
acb.ulACBLen = sizeof(ACB_MMIO);
acb.ulObjType = ACBTYPE_MMIO;
acb.hmmio = hmmio1;
/* Associate the waveform data object with the source handler */
if (rc = SpiAssociate(hstream1,hidSource1,&acb))
   return(rc);    /* error! */
/**********************************/
/***** CREATE MIDI STREAM (Stream #2)  **************/
/**********************************/
/* Get the stream handler ID for the File System Stream Handler */
if (rc = SpiGetHandler("FSSH",&hidSource2,&hidunused))
   return(rc);    /* error! */
/* Get the stream handler ID for the MIDI Audio Stream Handler */
if (rc = SpiGetHandler("AUDIOSH$",&hidunused,&hidTarget2))
   return(rc);    /* error! */
/* Create a data stream from the memory to an */
/* OEM audio device connected to speakers.   */
spcbkey.ulDataType = DATATYPE_MIDI;
spcbkey.ulDataSubType = SUBTYPE_NONE;
spcbkey.ulINtKey = 0;
strcpy(dcb.szDevName,"AUDIO99$");
dcb.ulDCBLen = sizeof(dcb);
dcb.hSysFileNum = (**hSysFileNum returned by audio dd on audio_init call)
```

```
 if (rc = SpiCreateStream(hidSource2,
                          hidTarget2,
                          &spcbkey,
                          NULL,
                          (PDCB)&dcb,
                          (PEVCB)&evcb2,
                          EventProc,
                          0,
                          &hstream2,
                          &hevent2))
    return(rc);    /* error! */
/* USE MMIO to access a waveform audio object */
 if ((hmmio2 = mmioOpen("E:Mygreat\midi\file\screams.mid"
                        NULL,
                        MMIO_READWRITE|
                        MMIO_DENYNONE)) == NULL)
    return(ERROR_FILE_NOT_FOUND);
/* Fill in acb with data object info */
 acb.ulACBLen = sizeof(ACB_MMIO);
 acb.ulObjType = ACBTYPE_MMIO;
 acb.hmmio = hmmio2;
/* Associate the waveform data object with the source handler */
 if (rc = SpiAssociate(hstream2,hidSource2,&acb))
    return(rc);    /* error! */
/*************************************************************/
/**** SETUP THE SYNCHRONIZATION BETWEEN THE TWO STREAMS ********/
/*************************************************************/
 slave[0].hSlaveStream = hstream2;  /* MIDI stream is slave */
 slave[0].mmtimeStart = 0;          /* Time to start slave */
 if (rc = SpiEnableSync(hstream1,       /* Audio stream is master */
                        &slave,
                        1,              /* Number of slaves */
                        0))             /* Use spcb sync granularity */
    return(rc);    /* error! */
/* Start streams by passing the handle to the stream sync master */
 if (rc = SpiStartStream(hstream1,SPI_START_SLAVES))
    return(rc);    /* error! */
/* Do other processing */
                  .
                  .
                  .
/* Create a semaphore and block this thread on the semaphore */
```

```
/* until the streaming is complete                          */
                  .
                  .
                  .
/* Destroy both streams when completed */
 if (rc = SpiDestroyStream(hstream1))
   return(rc);    /* error! */
 if (rc = SpiDestroyStream(hstream2))
   return(rc);    /* error! */
/* Perform other resource cleanup */
 }
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                  Event Procedure                         */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 RC EventProc(EVCB * pEVCB)
 {
 if (pEVCB == &EVCB1)    /* Stream #1 or Stream #2 event? */
   /* Process Stream #1 events */
 else
   /* Process Stream #2 events */
/* Handle events from the Sync/Stream Manager */
 switch(pEVCB->ulType) {
   case EVENT_IMPLICIT_TYPE:
       swich (pEVCB->ulSubType) {
          case EVENT_EOS:    /* End of Stream */
             /* Clear the semaphore so that the main routine can */
             /* continue execution.                              */
             break;
          case EVENT_ERROR:  /* Error occurred */
             /* flag error condition in main line application */
             break;
          case EVENT_STREAM_STOPPED:  /* Discard/Flush Stop */
             /* DO processing while stream stopped */
             break;
          case EVENT_SYNC_PREROLLED:  /* Stream is prerolled */
             /* Now that the producers are prerolled,do a real
                start to the stream handlers */
             break;
          default:
             /* Unknown event, do something */
          }
       break;
   default:
```

```
            / * Unknown event,do something */
        }
    }
```

图 3-3　同步化的 MIDI 和波形流

3.7　DLL 模型:文件系统流处理器

多媒体系统提供了两类流处理器,即在系统核心层(第 0 环)的 OS/ 2 PDD 和在应用程序层(第 3 环)的 OS/ 2 DLL。要有两类流处理器的原因是,一些流是由流处理器和设备的 PDD 之间的直接连接而理想控制着;而另一些流则未与映射特定设备的数据源或数据目标相联系。例如,文件流处理器是一个 DLL,因为所有文件系统 I/ O 功能可以通过第 3 环的 OS/ 2 API 来实现,并为所有文件系统设备服务。这样就不需要为文件系统可利用的每个设备都创建专门的流处理器 PDD。

3.7.1　文件系统流处理器模块

图 3-4 描述了文件系统流处理器(FSSHT.DLL)的简要结构。对图中的示范流处理器模块的介绍见表 3-3。

图 3-4　文件系统流处理器模块

表 3-3　文件系统流处理器示例模块

过程名	描　　述
SHINT .C	当装入 FSSHT .DLL 时获得控制,并执行以下函数: · DLL- INITTERM 初始化例程。包括向 SSM 注册流处理器,分配堆存储区,创建互斥号志。 · SHEXITLIST 终止例程。包括取消流处理器,关掉所有活动流,重装堆存储区,以及关掉互斥号志。
SHROUTER .C	从 SSM 接收流处理器命令(SHC)消息并分送给相应的低级例程。
FSSHASS .C	对 SHC- ASSOCIATE 流处理器命令消息进行处理,从而把一个目标(文件处理器,CD-ROM 驱动器号等)与一个数据流联系在一起。
SHEPROT .C	处理 SHC- ENUMERATE 流处理器命令消息,为指定的流返回一个流协议列表。
SHGPROT .C	对 SHC- GET- RROTOCOL 流处理器命令消息进行处理,为指定的数据类型返回一个流协议控制块(SPCB)。
SHIPROT .C	对 SHC- INSTALL- PROTOCOL 流处理器命令消息进行处理,为指定数据类型安装或重置一个 SPCB。
FSSHCREA .C	对 SHC- CREATE 流处理器命令消息进行处理,为指定的数据类型和子类型创建一个流事例。如果是为源流处理器创建,则在 FSSHREAD .C 中创建一个读线程。否则在 FSSHWRIT .C 中创建一个写线程。
SHNEGOT .C	对 SHC- NEGOTIATE- RESULT 流处理器命令消息进行处理,为一个指定的流事例保存协商后的 SPCB。
SHSTART .C	对 SHC- START 流处理器命令消息进行处理;为指定的流事例启动数据流。对于源流处理器,将执行 FSSHREAD .C 中的读线程;对目标流处理器,将执行 FSSHWRIT .C 中的写线程。
SHSTOP .C	对 SHC- STOP 流处理器命令消息进行处理,为指定的流事例停止数据流。对于源流处理器,所有缓冲区中的读线程要么废弃(STOP DISCARD),要么返回给 SSM(STOP FLUSH)。对于目标流处理器,将中止 FSSHWRIT .C 中的写线程,缓冲区将被废弃或刷新。
FSSHSEEK .C	对 SHC- SEEK 流处理器命令消息进行处理,查找流目标中的指定点。查找可以从文件头、当前位置或文件尾开始。可用偏移字节数或 MMTIME 单位指定查找点。该模块与 SEEKCALC .ASM 中的低级例程有接口,用于实现 MMTIME 值到字节的转换,同时它还与 mmioSeek 有接口,用于实现在目标中的实际查找。
SEEKCALC .ASM	实现 FSSHSEEK .C 中所要求的 MMTIME 到字节数以及字节数到 MMTIME 的转换。
SHDESTROY .C	对 SHC- DESTROY 流处理器命令消息进行处理,删除一个流事例。当接收到该消息后,将终止 FSSHREAD .C 中的读线程和 FSSHWRIT .C 中的写线程。

续表

过程名	描　　述
SHMISC .C	其它流处理器模块使用的支持例程。例如,查找流事件链以找到特定的流事例,查找一个扩充流协议块,或解除分配一个流事例。
FSSHREAD .C	从文件系统中读取一个目标。当在 FSSHCREA .C 中接收到 SHC-CREATE 消息时,将创建一个读线程,并在 SHDESTRY .C 中接收到 SHC-DESTROY 消息时删除该线程。读取操作是通过与 SHIOUTIL .C 中的 mmioRead 例程和低级例程的接口实现的,用于检验处理标志以及向 SSM 报告事件。
SHIOUTIL .C	FSSHREAD .C 和 FSSHWRIT .C 模块使用的低级例程。该模块实现对处理标志的检验并向 SSM 报告处理事件。
FSSHDAT .C	流处理器的全局数据说明。
makefile	创建 FSSHT .DLL 示例文件系统流处理器的 make 文件。

3 .7 .2　入口点图例

SSM 向更高级的 OS/ 2 多媒体器件(例如媒体驱动程序)提供 SPI 服务并输出 SSM 消息用以支持流处理器 DLL。通过标准的 OS/ 2 内部设备驱动程序通信(IDC)接口,SSM 把扩展的 SMH 消息输出给流处理器设备驱动程序。IDC 接口通过 DevHelp-AttachDO 函数建立。

图 3-5　DLL 流处理器结构

3 .7 .3　SMHEntryPoint

通过 SMHEntryPoint,设备驱动程序和 DLL 流处理器都能利用 SMH 消息。流处理器利用帮助器例程实现与 SSM 的注册,报告事件和同步尾接提示,以及请求或返回缓冲区。这些调用是同时的,第 3 环的 DLL 流处理器可把它们当作一个 DLL 调用来使用(对第 0 环的流处理器则作为一个 IDC 调用)。

3 .7 .4　SHCEntryPoint

正像图 3-6 所述,一个 DLL 流处理器通过一个附加的入口点实现其功能。DLL 流处理器必须提供一个接口以便与 SSM 进行通信。该接口称为流处理器命令(SHC)接口。

SSM 利用该接口输出的流处理器命令(SHC)来创建和控制数据流。这些 SHC 可以通过一个单独的入口点 SHCEntryPoint 来得到。

```
RC ShcRouter(pshc)
PSHC_COMMON pshcNow;

{/ * Start of ShcRouter */
RC rc = NO_ERROR;                    / * local return code */
   switch (pshc - > ulFunction)
   {
     case SHC_ASSOCIATE:
          {
            rc = ShcAssociate((PPARM_ASSOC)pshcNow);
            break;
          }
     case SHC_CREATE:
          {
            rc = ShcCreate((PPARM_CREATE)pshcNow);
            break;
          }
     case SHC_DESTROY:
          {
            rc = ShcDestroy((PPARM_DESTROY)pshcNow);
            break;
          }
     case SHC_SEEK:
          {
            rc = ShcSeek((PPARM_SEEK)pshcNow);
            break;
          }
     case SHC_START:
          {
            rc = ShcStart((PPARM_START)pshcNow);
            break;
          }
     case SHC_STOP:
          {
            rc = ShcStop((PPARM_STOP)pshcNow);
            break;
          }
```

```
        case SHC_GET_PROTOCOL:
            {
              rc = ShcGetProtocol((PPARM_GPROT)pshcNow);
              break;
            }
        case SHC_INSTALL_PROTOCOL:
            {
              rc = ShcInstallProtocol((PPARM_INSTPROT)pshcNow);
              break;
            }
        case SHC_ENUMERATE_PROTOCOLS:
            {
              rc = ShcEnumerateProtocols((PPARM_ENUMPROT)pshcNow);
              break;
            }
        case SHC_NEGOTIATE_RESULT:
            {
              rc = ShcNegotiateResult((PPARM_NEGOTIATE)pshcNow);
              break;
            }
        default:
            {
              rc = ERROR_INVALID_FUNCTION;
              break;
            }
      }/ * endswitch */

      return(rc);
    }/ * End of ShcRouter */
```

图 3-6　SHROUTER.C 文件中的 SHC 消息

3.7.5　DLL 初始化

图 3-7 描述了 DLL 流处理器的初始化过程。在初始化过程中,流处理器必须在初次装入 DLL 时完成所有全局 DLL 的初始化工作,并完成每个流处理器事例的初始化工作。DLL 的每个事例代表一个进程。在全局初始化过程中,流处理器必须向 SSM 注册。如果终止 DLL 需要的话,流处理器还应注册一个退出表例程。

```
    ULONG_DLL_InitTerm(HMODULE hmod,ULONG fTerm)

    {/ * Start of ShInit */
```

```
RC rc = NO_ERROR;                    /* local return code */
PARM_REG smhRegParm;                 /* Parameters for SMH_REGISTER */
int Registered = FALSE;
int HeapAllocated_Attached = FALSE;
int GlobDataMtxCreated = FALSE;
hmod;

  /*
   * Checking this parameter will insure that this routine will only
   * be run on an actual initialization. Return success from the
   * termination.
   */

  if (fTerm)
     {
     return(1L);
     }

  if (_CRT_init())
     {
     return(0L);
     }
  /*
   * Get the semaphore controlling the process count and update the process
   * count if successful.Then after we have the sem, if the count is 1,
   * we are guaranteed that we must do the global initializations.
   *
   * There is a problem. To determine if we need to create or open the
   * semaphore,we need to check the ProcessCount to see if this is the
   * first process.But since we don't have the semaphore,we don't have
   * a guarantee the count won't change. If we get caught in this window
   * then we will fail to get the semaphore and the rc will indicate error.
   */

if (ulProcessCount == 0)
  {/* First process */
    if (!(rc = DosCreateMutexSem(pszProcCntMutexName,
                                 &hmtxProcCnt,
                                 0L,
                                 TRUE)))
       {
         ulProcessCount++;
       }
  }/* First process */
```

```
else
  {/ * not first process */
   / * if a process exits and decrements ProcessCount before we get */
   / * the semaphore here,we will fail .*/
  if ( ! (rc = DosOpenMutexSem(pszProcCntMutexName,
                               & hmtxProcCnt)))
     {
       if ( ! (rc = DosRequestMutexSem(hmtxProcCnt,
                                       SEM_INDEFINITE_WAIT)))
         {
           ulProcessCount + + ;
         }
       else
         {
           DosCloseMutexSem(hmtxProcCnt);
         }
       }
  }/ * not first process */

if ( ! rc)
  {/ * Semaphore ok, In critical section */
   / *
     * If this is the first time this init routine is called then we are
     * being brought in by the loader the first time, so we need to
     * register with the SSM .
     */
    if (ulProcessCount = = 1)
       {/ * First process */
       smhRegParm .ulFunction = SMH_REGISTER;
       smhRegParm .pszSHName = pszHandlerName;
       smhRegParm .phidSrc = &SrcHandlerID;
       smhRegParm .phidTgt = &TgtHandlerID;
       smhRegParm .pshcfnEntry = (PSHCFN)ShcRouter;

       smhRegParm .ulFlags = 0L;
       if (ulHandlerFlags & HANDLER_CAN_BE_SOURCE)
          {
            smhRegParm .ulFlags = REGISTER_SRC_HNDLR;
          }
       if (ulHandlerFlags & HANDLER_CAN_BE_TARGET)
          {
            smhRegParm .ulFlags| = REGISTER_TGT_HNDLR;
          }
```

```
      if (ulHandlerFlags & HANDLER_IS_NONSTREAMING)
         {
           smhRegParm.ulFlags| = REGISTER_NONSTREAMING;
         }
      rc =  SMHEntryPoint((PVOID)&smhRegParm);
     /*
       * If ok then allocate our memory pool(heap),since it is under
       * semaphore control create and get the semaphore first .
       */
      if ( ! rc)
        {/ * Register ok */
          Registered = TRUE;
          hHeap = HhpCreateHeap(HEAPSIZE,HH_SHARED);

          if (hHeap)
            { / * Heap Allocated */
              HeapAllocated_Attached = TRUE;
              if ( ! (rc = DosCreateMutexSem(NULL,
                                                   &hmtxGlobalData,
                                                   DC_SEM_SHARED,
                                                   FALSE)))
                 {
                   GlobDataMtxCreated = TRUE;
                 }
              } / * Heap Allocated */
            else
              { / * Heap not allocated */
                rc  = ERROR_ALLOC_RESOURCES;
              } / * Heap not allocated */
            } / * Register ok */
       } / * First Process */
   else
      {/ * Not first process */
        if ( ! (rc  = DosOpenMutexSem(NULL,
                                        &hmtxGlobalData)))
          { / * Global data semaphore opened */

          GlobDataMtxCreated = TRUE;

          if ( ! ENTERCRIT(rc))
            {/ * Global Data Sem obtained */

              if (HhpAccessHeap(hHeap,HhpGetPID()))
                 {/ * Error accessing heap */
```

```
            rc = ERROR_ALLOC_RESOURCES;
          }/ * Error accessing heap */
        else
          {/ * Heap attached */
           HeapAllocated_Attached = TRUE;
          }/ * Heap attached */
        EXITCRIT;

      }/ * Global Data Sem obtained */
    }/ * Global data semaphore opened */
  }/ * Not first process */
/ *
 * If things are ok, Register an exit list handler and if that works
 * increment the process count .
 */
if ( ! rc)
  {
    rc = DosExitList(EXTLSTADD_ORDER,
                     (PFNEXITLIST)ShExitList);
  }
if (rc)
  {/ * Error occurred-Clean Up */
    if (HeapAllocated_Attached)
      {
        HhpReleaseHeap(hHeap,
                       HhpGetPID());
      }
    if (GlobDataMtxCreated)
      {
        DosCloseMutexSem(hmtxGlobalData);
      }
    if (Registered)
      {/ * Deregister */
       PARM_DEREG smhDeregParm;

       smhDeregParm.ulFunction = SMH_DEREGISTER;
       smhDeregParm.pszSHName = pszHandlerName;
       SMHEntryPoint(&smhDeregParm);
      }/ * Deregister */
    ulProcessCount--;
    DosReleaseMutexSem(hmtxProcCnt);
    DosCloseMutexSem(hmtxProcCnt);
```

```
        }/ * Error occurred-Clean Up */
     else
       {
   # ifdef MMRAS_PTRACE
           InitPTrace()
   # endif
           DosReleaseMutexSem(hmtxProcCnt);
       }

     }/ * Semaphore ok, In critical section */
    / *
      *  The return codes expected are:
      *  TRUE (any non-zero)-init routine worked
      * FALSE(zero)        -init routine failed
      *
      * So we have to reverse the local rc before we pass it back .
      */
     return( ! rc);

   }/ * End of SHInit */
```

图 3-7　DLL 流处理器初始化过程的伪代码

DLL **流处理器和设备驱动程序流处理器中的流事件处理逻辑基本上是一样的。只有一个显著的差别,即在** DLL **中流事件是在任务时间检测和报告的,而在设备驱动程序中是在中断时间。**

```
   # include     "os2 .h"
   # include     "os2me .h"

     RC            rc;                 / * Error return code              */
     HID           hidSource;          / * Source handler ID              */
     TIME_EVCB     timeevcb;           / * Cuepoint event control block   */
     HEVENT        hevent;             / * time event handle              */
     HSTREAM       hstream;            / * Stream handle                  */
     HID           hid;                / * Handler ID                     */
     PARM_EVENT    parm_event;         / * Report event parameter block   */
     MMTIME        mmtimeCurrent;      / * Current stream time            */
     PSMHFN        SMHEntryPoint;      / * Pointer to SMH entry point     */

   / * -----------------------------------------------------------------------*/
   / * Report an event .                                                       */
   / * -----------------------------------------------------------------------*/
     parm_event .ulFunction = SMH_REPORTEVENT;     / * Set function           */
```

```
    parm.event.hid = hidSource;                   /* Source handler ID        */
    parm.event.hevent = hevent;                   /* Event handle             */
    parm.event.pevcbEvent = (PEVCB)&timeevcb;     /* Pointer to Time EVCB     */

    timeevcb.ulType = EVENT.CUE.TIME;             /* Set event type           */
    timeevcb.hstream = hstream;                   /* Set stream handle        */
    timeevcb.hid = hid;                           /* Set handler ID           */
    timeevcb.ulStatus = 0;                        /* No status                */
    timeevcb.mmtimeStream = mmtimeCurrent;        /* Set current time         */

    if (rc = SMHEntryPoint(&parm.event))
    return(rc);                                   /* Error! */
```

图 3-8　DLL 流处理器事件检测

3.7.6　同步

DLL 流处理器可以把同步脉冲当作一个主体流或从体流来处理。同步组的主体必须设置 SPCB 中的 SPCBHAND.GENSYNC 或 SPCBHAND.NOSYNC 位,在 SHC.CREATE 调用时它们将返回给 SSM。否则,该流处理器不能作为主体。

如果想要创建一个能产生同步脉冲的流处理器,你必须利用硬件设备(例如声卡)产生固定速率的定时中断,或是利用 OS/2 的定时器服务,比如 OS/2 设备驱动程序的 SetTimer DlevHelp 或 OS/2 的 DosAsyncTimer 函数。然后你可以调用 GetDOSVar 函数查询全局信息分段结构(GlobalInfoSeg)。这使你能得到实时间,以检测丢失的时间信息。关于进一步的信息,参阅《OS/2 2.0 技术库物理设备驱动程序指南》。

注意:我们提供了用于同步的结构。音频流处理器可以作为主体(产生同步脉冲);也可以作为从体,但此时不支持对主体流的重新同步。

```
/*********************************************************************/
/* Master                                                            */
/* Synchronization-If we are the master stream, then report the      */
/*                 current time to the Sync/Stream Manager so that   */
/*                 it can be distributed to any slave streams.       */
/************** END OF SPECIFICATIONS ***************                */
PARM.EVENT parmevent;            /* parms for SMH.NOTIFY calls    */
SYNC.EVCB  syncevcb;             /* Cuepoint event control block  */

if (psib->ulStatus & STREAM.MASTER.SYNC) {
  parm.event.ulFunction = SMH.REPORTEVENT;    /* Set function             */
  parm.event.hid = psib->HandlerID;           /* Source handler ID        */
  parm.event.hevent = hevent;                 /* Event handle             */
  parm.event.pevcbEvent = (PEVCB)&syncevcb;   /* Pointer to Sync EVCB     */
  syncevcb.ulType = EVENT.CUE.TIME;           /* Set event type           */
```

```
    syncevcb.hstream = psib->hStream;              /* Set stream handle       */
    syncevcb.hid = hid;                            /* Set handler ID          */
    syncevcb.ulStatus = 0;                         /* No status               */
    syncevcb.mmtimeStream = mmtimeCurrent;         /* Set current time        */

    if (rc = SMHEntryPoint(&parm.event))
      return(rc)                                   /* Error! */
    }

 /*************************************************************************/
 /* Slave                                                                 */
 /* Synchronization-If we are the slave stream,then update the slave      */
 /*                  time and determine if this slave stream is           */
 /*                  too slow or too fast with respect to the master      */
 /*                  stream.                                              */
 /*                                                                       */
 /****************END OF SPECIFICATIONS*************                      */

 if ((pSTREAM)->ulStateFlg & STREAM_SLAVE_SYNC) {
        (pSTREAM)->SyncEvcb.mmtimeSlave = mmtimeCurrStreamTime;
        if ((pSTREAM)->SyncEvcb.ulStatus & SYNCPOLLING) {
             if (mmtimeCurrStreamTime > (pSTREAM)->SyncEvcb.mmtimeMaster)
                     /* I need to slow my stream */
                     ;
             if (mmtimeCurrStreamTime < (pSTREAM)->SyncEvcb.mmtimeMaster)
                     /* I need to speed up my stream */
                     ;
        }
```

图 3-9　DLL 流处理器同步

3.7.7　创建员工线程

流处理器 DLL 为每个产生的流创建员工线程。例如,员工线程可以包含以下代码:从 SSM 请求缓冲区,用设备数据进行填充,然后把它返回给 SSM。该过程可以一直持续下去,直到流尾或其它种类的流停止时。

一个员工线程独立完成流处理器的所有工作。为每个流事例创建一个员工线程是个不错的主意。一个员工线程基本上是在读例程或写例程内循环,这取决于它是源还是目标。如果是源,线程在读例程内循环,并可用它和设备接口中的任何命令从设备读取数据。它和 SSM 之间也有接口,用于请求空缓冲区和返回满缓冲区。另一方面,如果 DLL 流处理器是目标,则线程也是一个大循环,但执行不同的操作。它将向 SSM 请求满缓冲区并将它们顺利传送给设备 C 通过与设备接口中的任何方式来完成。如图 3-10 示。

```
#include <os2.h>
#include <os2me.h>
#include <hhpheap.h>
#include <shi.h>

PSIB psib;    /* Stream Instance Block */

{/* Start of FsshRead */

RC              rc = NO_ERROR;         /* local return code */
char            NeedBuf;               /* local loop boolean */
LONG            NumBytesIO;            /* Number of bytes from mmio */
PARM_NOTIFY     npget,npret;           /* parms for SMH_NOTIFY calls */
SRCBUFTAB       SrcBufTab = {0};       /* Source buffer table */
ULONG           ulStopped = DO_SLEEP;  /* did we get stop disc or flush */
BOOL            bAtEOS = FALSE;        /* End of Stream indicator */
ULONG           ulPostCount;           /* Temp to hold count */

  /* Before we start lets do some init stuff: */

   npget.ulFunction = SMH_NOTIFY;
   npget.hid = psib->HandlerID;
   npget.hstream = psib->hStream;
   npget.ulGetNumEntries = 1L;
   npget.ulRetNumEntries = 0L;
   npget.pGetBufTab = &SrcBufTab;
   npget.pRetBufTab = NULL;

   npret.ulFunction = SMH_NOTIFY;
   npret.hid = psib->HandlerID;
   npret.hstream = psib->hStream;
   npret.ulFlags = BUF_RETURNFULL;
   npret.ulGetNumEntries = 0L;
   npret.ulRetNumEntries = 1L;
   npret.pGetBufTab = NULL;
   npret.pRetBufTab = &SrcBufTab;

  /* Wait until we get the ShcStart */

   DosWaitEventSem(psib->hevStop,SEM_INDEFINITE_WAIT);

  /* We will loop forever getting an empty buffer,calling the device to */
  /* fill up the buffer, sending it to the consumer. During each        */
  /* iteration of the loop we will check the action flags for           */
  /* asynchronous requests to do things.                                */
```

```
if (psib - > ulActionFlags & SIBACTFLG_KILL)
   {/ *  Must have been a create error */
     rc = 1L;
   }/ * Must have been a create error */

/ *  Start the main loop */


while( ! rc)
  {/ * while no severe error */

  if (psib - > ulActionFlags)
    rc = CheckNSleep(psib);

 / *
   * Get a buffer
   */
  NeedBuf = TRUE;
  while((NeedBuf) & & ( ! rc))
    {/ * while we don't have a buffer */

    / * Clear the stop sem, so if after we call ssm to get a buffer if */
    / * it returns none avail then we won't miss a SSMBuffer Start     */
    / * before we go to sleep .                                         */
       DosResetEventSem(psib - > hevStop, & ulPostCount);

       npget .ulFlags = BUF_GETEMPTY;
       rc = SMHEntryPoint( & npget);/ * get a buffer */
       if ( ! rc)
          {
            NeedBuf = FALSE;
          / * make sure attribute is 0 so we don't pass around a bad value */
              SrcBufTab .ulMessageParm = 0L;
          }
       else
          {/ * return code from smhnotify */
          if (rc = = ERROR_BUFFER_NOT_AVAILABLE)
             {/ * buffer not available */

         / * the smhnotify resets the num entries to 0 when none avail */
             npget .ulGetNumEntries = 1L;

             ulStopped = DO_SLEEP;
             rc = SleepNCheck(psib, & ulStopped);
```

```
        }/ * buffer not available */
      }/ * return code from smhnotify */
    }/ * while we don't have a buffer */

  / * We have a buffer or an error */

  if ( ! rc)
    {/ * have a buffer-do the read */
      NumBytesIO = mmioRead((HMMIO)psib - > ulAssocP1,
                                (PCHAR)SrcBufTab.pBuffer,
                                (LONG)SrcBufTab.ulLength);
      if (NumBytesIO = = - 1L)
        {/ * an error */

          SrcBufTab.ulLength = 0L;
          / * get the real error code */
          rc = mmioGetLastError((HMMIO)psib - > ulAssocP1);

          rc = ShIOError(psib,npret,rc);

        }/ * an error */
      else
        {/ * We have some data */

          if (NumBytesIO ! = (LONG)SrcBufTab.ulLength)
            {/ * End of stream */
              npret.ulFlags | = BUF_EOS;
              bAtEOS = TRUE;
              DosResetEventSem(psib - > hevStop, &ulPostCount);
              SrcBufTab.ulLength = NumBytesIO;
            }/ * End of stream */

          / * Send the data to the stream manager */

          rc = SMHEntryPoint( &npret);
          if( ! rc)
            {/ * data sent ok */
              if (bAtEOS)
                {
                  bAtEOS = FALSE;
                  ulStopped = DO_SLEEP;
                  rc = SleepNCheck(psib, &ulStopped);
                }
              }/ * data sent ok */
            }/ * We have some data */
```

```
            /* Clear the EOS if it was set And attribute */
            npret.ulFlags = BUF_RETURNFULL;
            SrcBufTab.ulMessageParm = 0L;

          } /* have a buffer-do the read */

        } /* while no severe error */

      /* We get here if an error has occurred or a kill has */
      /* been sent.In the case of the kill, reset the */
      /* return code to 0 (no error)and exit the thread.*/
      /* Otherwise,report the error event and exit the */
      /* thread.*/
      if (psib->ulActionFlags & SIBACTFLG_KILL)
        {
          rc = 0L;
        }
      else
        {
         ReportEvent(psib,
                     rc,                  /* Return code */
                     EVENT_ERROR,         /* event type */
                     0L,                  /* user info */
                     NONRECOVERABLE_ERROR);/* Severe Error */
        }
    /* Only set this flag when we no longer need access to the sib since */
    /* Destroy may get control and Free the sib.*/

    psib->ulActionFlags |= SIBACTFLG_THREAD_DEAD;
    return;

  } /* End of FsshRead */
```

图 3-10 FSSHREAD.C

3.8 设备驱动程序模型:视频 PDD

设备驱动程序流处理器被作为第 0 环层的一个源处理器来使用,用于获得或返回数据缓冲区。该处理器还与硬件的物理设备驱动程序(PDD)有接口,用于接收或传送数据缓冲区指针。流处理器的目的是减轻媒体驱动程序向 PDD 传送数据流的繁重任务。另一方面,媒体驱动程序向 SSM 调用函数以初始化流。而 SSM 将要求流处理器在不受应用程序干预的情况下管理正确的流流动。如图 3-11 示。

设备驱动程序流处理器包括两个主要的入口点——SMHEntryPoint 和 DDCMDEn-

图 3-11　设备驱动程序流处理器结构

tryPoint。

3.8.1　SMHEntryPoint

通过 SMHEntryPoint,设备驱动程序流处理器将流管理帮助器(SMH)例程传送给 SSM。利用这些例程,流处理器向 SSM 注册,报告事件和同步提示,并请求或返回缓冲区。该接口是通过标准内部设备驱动程序通信(IDC)的方法建立的,由初始化过程中的 AttachDD DevHelp 函数来实现。

3.8.2　DDCMDEntryPoint

设备驱动程序流处理器通过 DDCMDEntryPoint 与硬件 PDD 进行通信。这可以利用设备驱动程序命令(DDCMD)来完成,它使流处理器可以要求 PDD 去执行诸如启动,停止或重启设备的功能。该接口利用了 ATTACHDD DevHelp 函数提供的 IDC 技术。处理器必须附于设备驱动程序上,以便获得设备驱动程序的 DDCMD 入口点地址。这一过程在发出设备驱动程序命令之前完成。

3.8.3　SHCEntryPoint

正如图 3-11 所示,设备驱动程序通过两个附加的入口点来扩充功能:SHCEntryPoint 和 SHDEntryPoint。

设备驱动程序流处理器接收 SSM 命令来初始化和执行流动功能。可以通过单独的一个入口点 SHCEntryPoint 获得这些流处理器命令(SHC)。主例程是一个与 SSM 设备驱动程序的 IDC 接口。通过利用诸如 SpiCreateStream, SpiStartStream 和 SpiStopStream 等流编程接口(SPI)函数,SSM 可以调用设备驱动器流处理器。

图 3-12 是 SHCEntryPoint 的实现代码。

```
RC DDCMDEntryPoint(PDDCMDCOMMON pCommon);/ * PDD entry point from SH */
RC SHDEntryPoint(PSHD_COMMON pCommon);  / * SH entry point from PDD */
RC SHCEntryPoint(PSHC_COMMON pCommon);  / * SH entry point from SSM */
RC SMHEntryPoint(PSHC_COMMON pCommon);  / * SSM entry point from SH */

ULONG ( * ShcFuncs[])(PVOID pCommon) = {  / * SHC function jump table */
            SHCAssociate,                / * 0 */
```

```
                SHCClose,                    /*1*/
                SHCCreate,                   /*2*/
                SHCDestroy,                  /*3*/
                SHCStart,                    /*4*/
                SHCStop,                     /*5*/
                SHCSeek,                     /*6*/
                SHCEnableEvent,              /*7*/
                SHCDisableEvent,             /*8*/
                SHCEnableSync,               /*9*/
                SHCDisableSync,              /*10*/
                SHCGetTime,                  /*11*/
                SHCGetProtocol,              /*12*/
                SHCInstallProtocol,          /*13*/
                SHCEnumerateProtocols,       /*14*/
                SHCNegotiateResult           /*15*/
                };
USHORT MaxShcFuncs = sizeof(ShcFuncs)/sizeof(USHORT);

/************************************/

RC    SHCEntryPoint(pCommon)
PSHC_COMMON    pCommon;
{
      if (pCommon->ulFunction > (ULONG)MaxShcFuncs)
              return(ERROR_INVALID_FUNCTION);
                                     /* Check for valid function */

return(ShcFuncs[pCommon->ulFunction](pCommon));
}                                    /* Call SHC message        */

/************************************/

RC SHCStart(PPARM_START pStart)
{
   PSTREAM            pSTREAM;           /* Ptr to current stream */
   DDCMDREADWRITE     DDCMDReadWrite;
   DDCMDCONTROL       DDCMDControl;

   ulRC               rc;                /* Return code           */

   if (rc = GetStreamEntry(&pSTREAM,pStart->hstream))
            return(rc);
   EnterCritSec;                         /* Disable interrupts */
switch(pSTREAM->ulStateFlg){
  case STREAM_RUNNING:
```

```
        /**********************************/
        /* Get a full buffer for playback */
        /**********************************/
        ParmNotify.ulFunction = SMH_NOTIFY;
        ParmNotify.hid = pSTREAM->hid;
        ParmNotify.hstream = pSTREAM->hStream;
        ParmNotify.ulGetNumEntries = 1;
        ParmNotify.ulRetNumEntries = 0;
        ParmNotify.pGetBufTab = &(pSTREAM->BufTab.aBufTab[usBufTabIndex]);
        ParmNotify.pRetBufTab = NULL;
        ParmNotify.ulFlags = BUF_GETFULL;
        SMHEntryPoint(&ParmNotify);

        /*********************************/
        /* Send full buffer to device (PDD) */
        /*********************************/

        DDCMDReadWrite.hStream = (*ppSTREAM)->hStream;
        DDCMDReadWrite.ulFunction = DDCMD_WRITE;
        DDCMDReadWrite.pBuffer = pCurrentBufTab->pBuffer;
        DDCMDReadWrite.ulBufferSize = pCurrentBufTab->ulLength;
        rc = (*pSTREAM->pDDCMDEntryPoint)(&DDCMDReadWrite);

        /**********************************/
        /* Start the device               */
        /**********************************/

        DDCMDControl.ulFunction = DDCMD_CONTROL;
        DDCMDControl.hStream = pSTREAM->hStream;
        DDCMDControl.pParm = NULL;
        DDCMDControl.ulParmSize = NULL;
        DDCMDControl.ulCmd = DDCMD_START;
        rc = (*pSTREAM->pDDCMDEntryPoint)(&DDCMDControl);
        break;

    }
    ExitCritSec;
        return(rc);
}
```

图 3-12 设备驱动程序流处理器 SHCEntryPoint

3.8.4 SHDEntryPoint

设备驱动程序流处理器从 PDD **接收命令来报告事件和中断。这些流处理器设备**

(SHD)帮助器命令是通过 SHDEntryPoint 提供的。该入口点特别用于 PDD 回调流处理器。例如 PDD 可以传送 SHD_REPORT_INT 命令来报告状态,表明缓冲区已满或表明需要一个附加的缓冲区。

图 3-13 是 SHDEntryPoint 的实现代码。

```
ULONG  ( * ShdFuncs[])(PVOID pCommon) = {/ * SHD message jump table */
                          SHDReportInt,              / * 0 */
                          SHDReportEvent             / * 1 */
                          };

USHORT MaxShdFuncs = sizeof(ShdFuncs)/ sizeof(USHORT);
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

RC SHDEntryPoint(pCommon)
PSHD_COMMON    pCommon;
{
     if (pCommon - > ulFunction > (ULONG)MaxShdFuncs)

            return(ERROR_INVALID_FUNCTION);/ * Check for valid function */
     return(ShdFuncs[pCommon - > ulFunction](pCommon));
     }                                  / * Call SHC message         */

/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

图 3-13　设备驱动程序流处理器 SHDEntryPoint

3.8.5　事件检测

设备驱动程序流处理器必须知道检测事件的时间。时间安排是重要的,要知道在什么时候启动和停止流,以及在什么时候报告特定的尾接提示点时间事件。有两种方法可以实现事件检测,即:

· 要求 PDD 监测时间;

· 利用一种随时监测实时间的算法。

3.8.5.1　PDD 监测时间

第一种也是最准确的一种事件检测的方法是从实际物理设备获得时间。通过利用 DDCMD_STATUS 从物理设备驱动程序请求当前流时间,这种方法可以实现。当应用程序请求一个事件时,流处理器向 PDD 发送 DDCMD_STATUS 命令来检测事件时间(附加的 DDCMD 消息允许 PDD 把时间通知给流处理器。详细介绍参见《OS/2 多媒体编程指南》一书)。当检测到时间后,PDD 通过 SHDEntryPoint 的 SHD_REPORT_EVENT 消息回应流处理器。此时,流处理器通过查表判断应该来的是哪一个事件。随后,通过

SMM_REPORTEVENT 的 SMHEntryPoint,流处理器向 SSM 报告该事件。一旦收到事件,SSM 把它回报给应用程序,如图 3-14 所示。

```
RC SHDReportEvent(PSHD_REPORTEVENT pRptEvent)
{
  PSTREAM             pSTREAM;
  ulRC                rc;

  if (rc = GetStreamEntry( &pSTREAM,pRptEvent - > hStream))
         return(rc);

  pSTREAM - > ulStreamTime = pRptEvent - > ulStreamTime;
 / * Update stream time */

 / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 / *  PDD detected an event and notified the stream handler        */
 / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  while (pEVENT ! = NULL) {
     if (pEVENT - > hEvent = = pRptEvent - > hEvent) {
                  RptEvent.ulFunction = SMH_REPORTEVENT;
                  RptEvent.hid = pSTREAM - > hid;
                  RptEvent.hevent = pEVENT - > hEvent;
                  RptEvent.pevcbEvent = &(pEVENT - > evcb);
                 / * * * * * * * * * * * * * * * * * * * * * * */
                 / * call SSM to report event arrival         */
                 / * * * * * * * * * * * * * * * * * * * * * * */
                  VideoSH.pSMHEntryPoint( &RptEvent);    / * report it */
                  break;    / * process only one event and break out */
     }
        pEVENT = pEVENT - > pNext;

     }/ * end while */

     return(NO_ERROR);
}
```

图 3-14　设备驱动程序流处理器事件检测

3.8.5.2　流处理器监测时间

事件检测的第二种方法是用流处理器监测时间,而不依赖于 PDD。与请求 PDD 监测时间的方法相比,这种方法不是特别准确和高效。它是用一种随时监测实时间的算法。当流处理器检测到合适的时间后,它向 SSM 通知事件,见图 3-15。

```
RC SHDReportEvent(PSHD_REPORTEVENT pRptEvent)
{
  PSTREAM              pSTREAM;
  ulRC                 rc;

  if (rc = GetStreamEntry( &pSTREAM,pRptEvent - > hStream))
          return(rc);

  pSTREAM - > ulStreamTime = pRptEvent - > ulStreamTime;
 / * Update stream time */

 / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 / *  Stream handler detected event and notified the SSM .            */
 / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  while (pEVENT ! = NULL) {                / * process all possible events */
     if (pEVENT - > hEvent = = pRptEvent - > hEvent) {
                   / * * * * * * * * * * * * * * * * * * * * * * */
                   / * Poll the current time to determine if the    */
                   / * event time is now .                          */
                   / * * * * * * * * * * * * * * * * * * * * * * */
                    if (mmtimeCurrTime > = pTimeEvcb - > mmtimeStream) {
                       pTimeEvcb - > mmtimeStream = mmtimeCurrTime;
                       RptEvent .ulFunction = SMH_REPORTEVENT;
                       RptEvent .hid = pSTREAM - > hid;
                       RptEvent .hevent = pEVENT - > hEvent;
                       RptEvent .pevcbEvent = &(pEVENT - > evcb);
                      / * * * * * * * * * * * * * * * * * * * * * * */
                      / * call the Sync/ Stream Manager to report      */
                      / *  event arrival .                             */
                      / * * * * * * * * * * * * * * * * * * * * * * */
                       VideoSH .pSMHEntryPoint(&RptEvent);/ * report it */
  }
    pEVENT = pEVENT - > pNext;
    }/ * end while */

    return(NO_ERROR);
}
```

图 3-15　设备驱动程序流处理器事件检测

需要注意的是,当你开发一个流处理器时,它应能支持所有时间格式。

注意:大多数物理设备驱动程序用毫秒作为时间参考值;但流请求的那些由应用程序初始化的尾接提示点是以 MMTIME 为时间格式的。因此,流处理器必须能实现 MMTIME 和毫秒之间的相互转换。

3.8.6 尾接提示点

尾接提示点与流是共存的,也就是说,一旦被允许,它们将在整个流的持续时间内一直存在,而不管用户决定向前还是向后查找。如果用户越过一个事件进行查找,则流处理器将检测不到该事件。如果用户向后查找,流处理器应该重新允许该事件。

3.8.7 错误检测

流处理器能检测错误并将它们报告给 SSM,以便送回给媒体驱动程序。一旦物理设备驱动程序检测到一个错误,PDD 必须通过 SHDEntryPoint 将它报告给流处理器。随后,流处理器可采取相应行动。如果是硬件错误,流处理器必须停止设备,并向 SSM 报告一个事件。流处理器通过提供一个错误码表明发生了一个硬错误。对一些常规事件,如过缓和过速,目标流处理器将试图得到另一个缓冲区,然后暂停设备。当更多的数据可用于输出后,目标流处理器将被重新启动。这种情况将造成输出数据流中的一个断点。如果使用交错数据,当已到达数据尾,但还未到文件尾时,就会产生过缓。

图 3-16 是流处理器检测错误的实现代码。

```
RptEvent.ulFunction = SMH_REPORTEVENT;
RptEvent.hid = pSTREAM->hid;
RptEvent.hevent = 0;                        /* must be 0 */
RptEvent.pevcbEvent = (PEVCB)&evcb;
evcb.ulType = EVENT_IMPLICIT_TYPE;          /* SPI event */
evcb.ulSubType = EVENT_ERROR;               /* event type */
evcb.ulFlags = 0;
evcb.ulStatus = EVENT_ERROR;                /* error code */
evcb.hStream = pSTREAM->hStream;
/************************/
/* call SSM to report event arrival    */
/************************/
VideoSH.pSMHEntryPoint(&Ramp.ptEvent);            /* report it! */
```

图 3-16 设备驱动程序流处理器错误检测

3.8.8 同步

每个流处理器不一定能产生或接收同步脉冲,这是由处理器的 SPCB 决定的。同步脉冲是作为主体流处理器发出的一个事件来传送的。

根据所编写的流的同步关系,SSM 分送同步脉冲。这对 DLL 和设备驱动程序流处理器都是适用的。设备驱动程序流处理器通过它们的同步脉冲 SYNC_EVCB 接收同步脉冲。每个从体流处理器必须用计算后的流时间随时更新同步脉冲 SYNC_EVCB。SSM 按主体流时间检查从体流时间,并决定是否给该流处理器发送一个同步脉冲。设备驱动程序流处理器通过轮询同步脉冲 SYNC_EVCB 中的一个标志来检测 SSM 发来的同步脉冲。SSM 通过设置该标志来指示一个同步脉冲并更新当前主体流时间。典型的情况是,在中断处理过程中 PDD 从体处理器轮询一次该标志,并相应调整流的使用情况。图 3-17 是一个设备驱动程序流处理器支持同步的示例。

```
/*************************************************************/
/* If we are the master, then report the time so that the slaves   */
/* can adjust their time and be in sync with the master.            */
/*************************************************************/

if ((pSTREAM)->ulStateFlg & STREAM_MASTER_SYNC) {
        (pSTREAM)->SyncEvcb.mmtimeMaster = mmtimeCurrStreamTime;
        RptEvent.ulFunction = SMH_REPORTEVENT;
        RptEvent.hid = (pSTREAM)->hid;
        RptEvent.hevent = 0;        /* must be 0 for implicit events */
        RptEvent.pevcbEvent = (PEVCB) &((pSTREAM)->SyncEvcb);
        VideoSH.pSMHEntryPoint(&RptEvent);/* report master time */
}
/************************************************************/
/* If we are the slave, then update slave time                     */
/* and determine if this slave stream is too slow or too fast      */
/* with respect to the master.                                     */
/************************************************************/

if((pSTREAM)->ulStateFlg & STREAM_SLAVE_SYNC) {
       (pSTREAM)->SyncEvcb.mmtimeSlave = mmtimeCurrStreamTime;
       if ((pSTREAM)->SyncEvcb.ulStatus & SYNCPOLLING) {
            if (mmtimeCurrStreamTime > (pSTREAM)->SyncEvcb.mmtimeMaster)
                   /* I need to slow my stream */
                   ;
            if (mmtimeCurrStreamTime < (pSTREAM)->SyncEvcb.mmtimeMaster)
                   /* I need to speed up my stream */
                   ;
       }
```

图 3-17　流处理器同步

注意:关于同步脉冲产生和处理的进一步介绍,参阅 3 .2 节“同步的特点”。

3 .9 内部设备驱动程序通信(IDC)

OS/ 2 多媒体音频设备驱动程序是一个标准的 OS/ 2 PDD,它利用两个 OS/ 2 接口定义了 IBM 音频设备驱动程序的标准:音频 IOCtl 接口和内部设备驱动程序通信(IDC)接口。这些接口构成了 OS/ 2 多媒体应用程序与媒体驱动程序共享音频设备的基础。

音频 IOCtl 接口是作为 OS/ 2 DosDevIOCtl 函数的子函数提供的。在第 3 环优先级上运行的多媒体驱动程序利用 IOCtl 接口来设置用于存取的音频设备以及改变卷控制等。IOCtl 接口并一定要用于向设备驱动程序传送音频数据。

IDC 接口由 OS/ 2 ATTACHDD DevHelp 函数提供。流处理器设备驱动程序和音频物理设备驱动程序利用 IDC 接口互相通信,以协调它们向音频设备流动数据的工作。

3 .9 .1 IDC 接口

为参与 IDC 接口的两个设备驱动程序提供了两个入口点:DDCMDEntryPoint 和 SHDEntryPoint。DDCMDEntryPoint 允许流处理器设备驱动程序与 PDD 进行通信,SHDEntryPoint 则允许音频 PDD 与流处理器进行通信。

音频 PDD 模块见图 3-18。需要注意的是,流数据缓冲区指针是 SSM 通过 SMH 调用传送给音频流处理器的。然后音频流处理器再通过 DDCMD 消息把这些指针传送给 PDD。

3 .9 .1 .1 与 PDD 通信

从 DDCMD 的入口点定义了 t 个消息。每个消息代表了流处理器要求 PDD 执行一个动作的特定请求。

读和写消息将允许流处理器从 PDD 直接获取数据,而不受应用程序的干预。应用程序不必通过 IOCtl 传送数据,也不用分配和锁定内存来进行数据传送。

注意:关于进一步信息,参阅《OS/ 2 多媒体编程指南》一书中的设备驱动程序命令(DDCMD)消息。

3 .9 .1 .2 与流处理器通信

SHDEntryPoint 包含下列两个消息,它们在 \ TOOLKIT \ H 子目录下的 SHDD .H 文件中。SHDD .H 中有数据结构及其类型定义,还有定义特定值的 # define 语句。需要注意的是,为了以后更大的灵活性,消息将把指针传送给数据包。

1 . SHD_REPORT_INT

当 PDD 在中断时有所需求时,它使用该消息。例如,它用该消息告诉流处理器:它已用完了所有数据,急需更多的数据。

当流处理器得到该信息后,它知道 PDD 正在用一个可能用过的缓冲区。于是流处理器作出回应,即给 PDD 一个新缓冲区供使用。

2 . SHD_REPORT_EVENT

流处理器利用该消息来与 PDD 的工作保持同步。例如,流处理器可能要求 PDD 每

图 3-18　音频物理设备驱动程序模块

隔 0.1 秒回报处理的数据。流处理器有处理这些事件的所有逻辑。PDD 检查请求,当它知道已对数据处理了 0.1 秒后,它调用 SHD_REPORT_EVENT。此时,流处理器可相应行事,PDD 则返回。

只有 PDD 真正知道该过程,换句话说,只有 PDD 知道有多少毫秒的数据已被设备处理过了。流处理器根据流过数据的多少进行计算,也大致知道处理过的数据。但是流处理器不能按毫秒来计算处理的数据,而 PDD 甚至可用比毫秒更小的时间。

3.9.2　流处理器值

流处理器要寻找指定的值。例如,当流处理器在 DDCMD_CONTROL 消息时请求停止或暂停,返回给流处理器的指针是一个指向 PDD 以实时记录的累加时间的指针。因此,不管在什么时候流处理器要求设备停止, PDD 将尊重这一要求并把 PDD 在流中停止的实时间告诉流处理器。

流处理器希望得到的另一个值是在 DDCMD_SETUP 消息时返回的。这也是一个指向 PDD 记录的累加时间的指针,但是指在流处理器请求时流被首次启动的时间。

3.9.3 PDD 值

在 DDCMD_SETUP 消息时,流处理器把一个指针传送给 PDD。PDD 用该指针值来设定 PDD 的参考时间。我们不希望在每次启动流时,PDD 都把它的时间设为 0,因为 PDD 也许已经在流中进行过一次查找了。也可能 PDD 已经处理了 1 分钟的数据,然后它又反向查找到可能存在于数据中的 30 秒标记。如果我们此时启动,我们不希望 PDD 认为它又从 0 时刻开始了。因为实际上它是从流中的 30 秒标记处开始的。

DDCMD_CONTROL 有一个重要的 NOTIFY 子函数,用于尾接提示点或事件检测。流处理器支持尾接提示点事件——当到达文件的一个特定位置或过了特定的一段时间后,应用程序要求被通知。流处理器用两种方法来检测已过的时间:

1.利用 DDCMD_CONTROL NOTIFY 子函数。在一个特定的时刻要求 PDD 通知流处理器并传送一个指向尾接提示点时间的指针。

2.流处理器在程序内部确定时间。这不如第一种方法精确,因为只有 PDD 知道实时间。

例如,流处理器利用了一个一分钟的 DDCMD_CONTROL NOTIFY。如果 PDD 支持精确事件检测,它必须接受该请求并将它放入队列中,最好是一个连接表。连接表将得到一分钟时间,这样在流动过程中,PDD 将在必要时检测是否处在一个一分钟标记处。如果该事件发生,PDD 回应一个 SHD_REPORT_EVENT。然后,程序员可以将事件检测连接表节点解开。

要牢记的是,PDD 应具有将请求排队的能力,因为可能会有附加的请求。例如,应用程序可能要求在 1 分钟标记处被通知,然后在一分半钟处,还可能在每 5 分钟处。

如果 PDD 不支持事件检测,则在一个 DDCMD_CONTROL NOTIFY 时,当要求它接收请求时,它回应 ERROR_INVALID_REQUEST。这告诉流处理器它必须自己检测事件。

注意:IDC 接口返回码可参阅《OS/2 多媒体编程指南》一书。

3.10 调整同步/流管理器工作

同步/流管理器(SSM)由两个模块组成:SSMDD。SYS 和 SYS.DLL。SSMDD.SYS 是 SSM 的第 0 环设备驱动程序。它的语法结构如下:

参数

/ **S**: sss	指定能同时创建的流数目。取值范围是 1—64。对于内存大于 8MB 的机器,默认值是 12;对于内存为 8MB 或更小的机器,默认值为 6。
/ **P**: ppp	指定能同时创建的进程数。取值范围为 1—64。对于内存大于 8MB 的机器,默认值为 12;对内存为 8MB 或更小的机器,默认值是 6。
/ **H**: hhh	指定使用的最大堆空间(KB)。取值范围为 16—256。默认值是 64。
/ **Q**: qqq	指定(每个进程)的最大事件队列。取值范围为 2—1024;默认值为 64。
/ **E**: eee	指定(每个流)可允许的事件数。取值范围为 1—1024。对于内存大于 8MB 的机器,默认值是 32;对于内存为 8MB 或更小的机器,默认值为 20。

注意: DEVICE = SSMDD .SYS 语句必须是 CONFIG .SYS 文件中的第一条第 0 环流处理器语句。

同步/ 流资源限制如下:

· 最大流数为 64。

· 一个同步组中的最大流数为 64。

· 控制流的最大进程数是 64。

· 每个进程的最大同步/ 流事件队列可容纳 1024 个事件。

第4章 I/O 过程

本章描述如何编写客户文件格式的I/O过程(IO Proc)。提供了位于\TOOLKIT\SAMPLES\MM子目录下的以下I/O过程的源代码:

1. Case_Converter

提供一个编写文件格式I/O过程(不描述数据翻译的应用)的简单例子。该实例执行文本的情况转换。参见\TOOLKIT\SAMPLES\CASECONV子目录。

2. M_Motion

提供一个编写与图象文件格式一起使用的I/O过程的例子。该实例使文件格式对M_Motion静态视频文件透明并描述了数据翻译的应用。参见\TOOLKIT\SAMPLES\MM\MMIOPROC子目录。

3. Ultimotion*

提供一个详细的为软件运动视频文件格式写I/O过程时所要考虑到的情况的例子。ULIOT包括了CODEC支持并描述了如何集成普通的和文件格式一定的代码以支持多功能I/O过程。参见\TOOLKIT\SAMPLES\MM\ULTIMOIO子目录。

以下各节讨论M_Motion I/O过程实例支持的消息,以及这些消息所要求的最小的处理。同时也提供了Ultimotion I/O过程的信息,描述如何调用及初始化一个CODEC过程。

4.1 I/O过程结构

OS/2多媒体的MMIO子系统使应用程序、媒体控制驱动程序和流处理器独立于数据一定的处理,其方式与媒体控制接口缓冲区应用程序与设备一定的处理相独立一样。应用程序通过MMIO管理器发送MMIO函数,MMIO管理器用I/O过程(IOProc)来处理特定类型的多媒体数据。图4-1描述了OS/2多媒体安装时所提供的过程。

4.1.1 消息处理

IOProc是一个基于消息的处理器。应用程序和MMIO子系统通过使用MMIO消息与IOProc通信。当MMIO收到应用程序的请求时,MMIO管理器为该操作发送给负责那个特定文件格式或存储系统的IOProc一条消息。于是,I/O过程执行基于其从MMIO管理器或应用程序接收到的消息的操作。

MMIO消息可以是预定义的,也可以是用户定义的。

1. 预定义消息

这是由MMIO管理器为其相关函数发送的系统消息。例如,发送mmioOpen函数的应用程序使MMIO管理器向I/O过程发送一条MMIOM_OPEN消息,以打开特定文件。

这些消息使得应用程序以独立格式的方式处理媒体文件。MMIO 管理器确定了处理该消息的正确的 I/O 过程,依据是一个 I/O 过程标识和函数中指定的 I/O 过程类型。参见 4.1.2 节"I/O 过程标识(FOURCC)"和 4.1.3 节"I/O 过程类型"。

图 4-1 支持的 I/O 过程

2. 用户定义消息

这是一条通过采用 mmioSendMessage 函数从应用程序直接送至 I/O 过程的私有消息。该函数使程序能够直接调用一个 I/O 过程(不像系统消息那样由 MMIO 管理器传送)。\TOOLKIT\H 子目录中的 MMIOOS2.H 定义了标识 MMIO_USER 使用户能够创建自己的消息。mmioSendMessage 函数要求客户消息的值定义得等于或高于 MMIOOS2.H 文件中定义的 MMIOM_USER 值。

4.1.2 I/O 过程标识符(FOURCC)

FOURCC 是一个 32 位的量,它代表 1 至 4 个 ASCII 字母数字字符组成的序列(右边填上空字符)。每个 I/O 过程支持一个特定的文件格式。文件格式和 IOProc 由特定的 FOURCC 代码表示。允许 FOURCC 当作 ID 值使用,而不是文件格式的国家语言支持(NLS)字符串名或文件名的扩展。

支持多种媒体类型的格式要求每个变量有一个不同的 FOURCC,作为每种媒体类型

的不同的 I/ O 过程。

注意: 如果没有提供四字符代码, 可以用 mmioIdentifyFile 函数标识。

4.1.3 I/ O 过程类型

不同的 MMIO 函数操作于特定的 IOProc 类型。有两种 I/ O 过程: 文件格式和存储系统。文件格式 IOProc 操作文件的内容, 需要时调用其它的 MMIO 服务。相反, 存储系统 IOProc 操作媒体对象的存储显示, 并调用基本的操作系统服务。

初始化过程中要指定 I/ O 过程类型, 可以为文件格式 IOProc 的 MMIO_IOPROC_FILEFORMAT 或存储系统 IOProcs 的 MMIO_IOPROC_STORAGE SYSTEM 设置 MMFORMATINFO 结构中的 ulIOProcType 字段。还必须有一个源文件, 以指定描述 I/ O过程的 NLS 名。如果 I/ O 过程是用于多种国家, 还应该将 RC 文件装订到 DLL 上。IOProc 必须处理 GETFORMATINFO 和 GETFORMATNAME 消息以提供上述信息。

4.1.3.1 文件格式 I/ O 过程

文件格式 IOProc 必须支持所有的 MMIO 系统消息(除了 RIFF 组合文件消息), 还必须处理所有由应用程序创建的用户定义的消息。例如, 一个文件格式的 IOProc 必须支持 MMIOM_GETFORMATINFO 消息, 因为 MMIO 管理器在安装时即从内部发送这条消息给 IOProc。如果不支持 MMIOM_GETFORMATINFO 消息, 则在 MMIO 内置 IOProc 表上为特定 IOProc 放置一个空白的 MMFORMATINFO 结构, FOURCC 除外。

另外, 默认的消息句柄还应支持系统消息, 这些消息 MMIO 不支持。消息句柄应将任何它不支持的消息传送给子过程 IOProc。例如, 图 4-2 描述了消息如何从一个文件格式 IOProc 传到一个存储系统 IOProc。

```
default:
    {
    /*
      * Declare Local Variables.
      */
     PMMFILESTATUS      pVidInfo;
     LONG               lRC;
    /*********************************
      * Check for valid MMIOINFO block.
      *********************************/
     if (! pmmioinfo)
        return (MMIO_ERROR);
    /*********************************
      * Set up our working variable MMFILESTATUS.
      *********************************/
     pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;
     if (pVidInfo != NULL && pVidInfo->hmmioSS)
```

```
    {
     lRC = mmioSendMessage (pVidInfo- > hmmioSS,
                             usMsg,
                             lParam1,
                             lParam2);
     if ( ! lRC)
        pmmioinfo- > ulErrorRet = mmioGetLastError (pVidInfo- > hmmioSS);
     return (lRC);
    }
else
    {
     if (pmmioinfo ! = NULL)
        pmmioinfo- > ulErrorRet = MMIOERR_UNSUPPORTED_MESSAGE;
     return (MMIOERR_UNSUPPORTED_MESSAGE);
    }
}  / * end case of Default */
```

图 4-2　默认消息句柄(MMOTPROC.C)

如果要编写支持翻译的客户 IOProc,下列消息需要考虑翻译标志:

- MMIOM_OPEN
- MMIOM_READ
- MMIOM_WRITE
- MMIOM_SEEK
- MMIOM_CLOSE
- MMIOM_GETHEADER
- MMIOM_SETHEADER
- MMIOM_QUERYHEADER

4.1.3.2　存储系统 I/O 过程

存储系统 IOProc 处理操作于存储系统的系统定义的消息的子集。例如,DOS 和 MEM 的 IOProcs 处理下列消息:

- MMIOM_OPEN
- MMIOM_READ
- MMIOM_WRITE
- MMIOM_SEEK
- MMIOM_CLOSE
- MMIOM_GETFORMATNAME
- MMIOM_GETFORMATINFO
- MMIOM_IDENTIFYFILE

指定给存储系统 IOProc 的消息,如文件的扩展属性等,将通过文件格式 IOProc 的默

认句柄传给存储系统 IOProc 处理。

注意：安装在 OS/2 多媒体上的 RIFF 组合文件(CF)IOProc 只支持 MMIOM_IDENTIFYFILE，MMIOM_GETFORMATINFO 和 MMIOM_GETFORMATNAME。组合文件 IOProc 不需要支持其它的 MMIO 消息，因为卷(BND)IOProc 执行直接的文件 I/O 操作。这两个 IOProc 可以作为一个逻辑组合文件 IOProc。

4.2 数据翻译和文件转换

MMIO 在其 API 中提供了一组选项，支持两种文件存取模式——翻译和不翻译。这些模式使应用程序在执行 I/O 操作时，以纯粹独有的格式或标准显示格式存取数据。IOProc 可以在编写时指定支持这两种存取方法。

默认存取模式——不翻译，允许调用者以原始格式执行文件数据的 I/O，所有的头信息以及所有数据都向文件写或从文件读并且不加修改地在调用层显示。

另一种存取模式——翻译，是一种用来隐藏独有数据格式的方式，并允许调用者为特定媒体类型(如音频、图象或 MIDI)使用标准头和数据格式。为此目的，以标准显示格式定义了一组标准头信息和相应的数据格式。IOProc 可以在编写时指定支持标准格式。它执行头信息、数据信息的翻译，或在读写类型操作中为该媒体类型执行从原始格式到标准格式的翻译。翻译是对文件头和数据执行的。

由于每对 IOProc 的强大的翻译能力，可以很容易地将文件从一种格式转换为另一种格式。文件转换仅仅是从一个文件调入一种向另一个文件存储的组合。例如，应用程序可以从一个 AVC 图象文件读，并能翻译成标准显示格式，即 OS/2 位图。应用程序可以根据需要应用该位图，包括将其显示在屏幕上，图象处理及打印。反过来，位图也能存成另一种文件格式如 M-Motion，方法是写入带翻译能力的 M-Motion IOProc。例如，I/O 过程和应用程序必须为每个媒体类型使用相同的标准显示格式，使得转换成为可能。(定义的媒体类型有图象、音频、MIDI、数字视频及电影。)数据转换如图 4-3 示。这些标准格式

图 4-3 数据转换

提供给媒体描述性(头)信息及媒体内容。标准描述结构(description structure)是每个文件格式通常使用的头的最高级集合。这允许所有的格式将其信息的子集放入标准格式，以方便其它应用程序存取。同样地，每个指定格式只能查询它所需要的子集。标准内容格式(content format)是一种有用的数据格式显示，它包括了尽可能高的质量。

翻译功能辅助这些标准格式保证数据在应用程序、IOProcs 及操作系统服务之间移动。包括描述性信息的结构具有可映射到系统结构的字段，如 OS/2 操作系统的 BITMAPINFOHEADER。内容格式必须直接可被操作系统和服务，或标准硬件设备

使用。

描述性头和内容格式应紧密耦合。如果文件包括一个媒体项,则应用程序能查询描述该媒体的头。IOProc 返回头,该头中包括了与文件中实际存在的信息最紧密配合的支持的内容格式。例如,如果一个图象文件包括 21 位 YUV 数据,则该文件的 IOProc 就通知应用程序它提供了 24 位 RGB。IOProc 有责任将所有后来的读操作从 YUV 翻译到 RGB。另外,当应用程序创建一个新的媒体元素时,它可以为一个新的媒体项设置头。所有后来的翻译的写操作,即从应用程序到 IOProc 的操作,都必须包括由该头描述的内容格式。

每种数据类型使用不同的描述结构和内容格式。表 4-1 给出了支持媒体类型的标准显示格式的总览。

表 4-1　标准显示格式

媒体	头	数　　据
音频	MMAUDIO	PCM 11.025, 22.05, 33.1 kHz
图象	MMIMAGE	OS/2 1.3 位图(24 位 RGB,1,4,8 位调色板)
MIDI	MMMIDI	格式 0 或 1
电影	MMMOVIE	多声道视频和音频
视频	MMVIDEO	16, 24 位 RGB, 4, 8 位调色板

注意: 组合多媒体文件中的数据翻译只对文件中的媒体元素而言。翻译不作用于非多媒体文件。

4.2.1　MMFORMATINFO 数据结构

几种 MMIO 函数在媒体转换时均采用了 MMFORMATINFO 数据结构。mmioOpen 函数包括在 MMIOINFO 结构中的 ulTranslate 字段定义的 MMIO_TRANSLATEHEADER 和 MMIO_TRANSLATEDATA 标志。所有后来的对多媒体文件的读写操作均返回基于这些标志的数据。目前,翻译只是为图象和音频定义的。MMIOOS2.H 头文件定义了如图 4-4 所示的 MMFORMATINFO 结构。

```
typedef struct_MMFORMATINFO {      /* mmformatinfo                    */
                                                                      */
   ULONG    ulStructLen;           /* Length of this structure         */
   FOURCC   fccIOProc;             /* IOProc identifier                */
   ULONG    ulIOProcType;          /* Type of IOProc                   */
   ULONG    ulMediaType;           /* Media type                       */
   ULONG    ulFlags;               /* IOProc capability flags          */
   CHAR     szDefaultFormatExt[sizeof(FOURCC) + 1];   */
                                   /* Default extension 4 + null       */
   ULONG    ulCodePage;            /* Code page                        */
   ULONG    ulLanguage;            /* Language                         */
```

```
    LONG      lNameLength;        /* Length of identifier string    */
    } MMFORMATINFO;
```

图 4-4 MMFORMATINFO 数据结构

4.3 I/O 过程入口指针

图 4-5 描述了用于存取 I/O 过程的功能的入口指针。

```
LONG APIENTRY IOProc_Entry ( PVOID     pmmioStr,
                             USHORT    usMessage,
                             LONG      lParam1,
                             LONG      lParam2)
```

图 4-5 I/O 过程入口指针

与其相关的参数如表 4-2 示。

表 4-2 与 I/O 过程入口指针相关的参数

参数	描　　述
PVOID pmmioStr	为一个包含打开文件信息的 MMIOINFO 数据结构指定一个指针。
USHORT usMsg	指定要求文件 I/O 过程处理的消息。(用户定义的消息必须在前面为消息定义成 MMIOM_USER。)
LONG lParam1	指定与消息有关的信息,如文件名。
LONG lParam2	指定其它与消息有关的信息。(有的消息将其用作值。)

注意: 返回值与消息有关。如果 I/O 过程不能识别一个由 usMsg 传入的消息,并且默认的消息句柄不能识别 usMsg,则 I/O 过程将返回 MMIOERR_UNSUPPORTED_MESSAGE。

4.4 支持的消息

图 4-6 描述了由 M-Motion I/O 过程(MMOTTK.DLL)支持的消息。图后的内容是文件格式 I/O 过程必须支持的消息的描述及代码示例。

4.4.1 MMIOM_OPEN

每个 IOProc 都应该能够处理 MMIOM_OPEN 消息,该消息要求打开一个文件。一旦应用程序知道哪个 IOProc 与所选文件相关,它便采用 mmioOpen 函数打开该文件。应用程序用标识处理过程所提供的 FOURCC 检验合适的 IOProc。

当 MMIO 管理器发出一个 MMIOM_OPEN 消息时,文件格式的 IOProc 必须检查下列项:

图 4-6　M-Motion I/ O 过程支持的消息

· 文件格式 IOProc 使用 pmmioinfo 结构的 fccChildIOProc 字段并执行另一个 mmioOpen。这种情况应设置 MMIO_NOIDENTIFY 标志。
· MMIOINFO 结构的 lLogicalFilePos 应设为 0 或,如果有头的话,在头之后的数据的第一位。该例有头,并且用 mmioSeek 的返回代码设置了 lLogicalFilePos。
· 文件格式 IOProc 必须检查是否设置了 MMIO_TRANSLATEDATA 或 MMIO_TRANSLATEHEADER 标志。如果设置了翻译标志,则它根据一组定义好的交互格式来处理数据(详细内容请参阅《OS/ 2 多媒体编程参考》(OS/ 2 Multimedia Programming Reference) 一书)。如果未设置翻译标志,则允许数据经过应用程序指定的修改通过 IOProc。如果 IOProc 必须被多媒体数据转换程序支持则要求支持翻译。

如果 OPEN 成功,则应用程序可以通过使用 mmioGetHeaderInfo 消息获得文件中媒体的信息。

图 4-7 描述了如何为文件格式 IOProc 处理 MMIOM_OPEN 消息。MMIOM_OPEN 消息句柄使用 mmioOpen 函数确定一个使用支持 MMIO 的存储系统 IOProc 的媒体数据对象。当打开该数据对象时,向文件格式 IOProc 返回一个 hmmio 句柄(H1)。该句柄存在文件格式 IOProc 的 MMIOINFO 的 aulInfo[1] 字段。当返回给应用程序发送的 mmioOpen 函数时,必须注意到已经产生了另一个句柄(H2)并且返回给了应用程序。这些句柄允许存取数据对象。应用程序将使用 H2,而文件格式 IOProc 将使用带有向存储

系统 IOProc 调用 MMIO 函数的 H1。

图 4-7 所示的例子显示了 M-Motion IOProc 是如何支持 MMIOM_OPEN 消息的。

```
case MMIOM_OPEN:
    {
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Declare local variables
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    PMMFILESTATUS     pVidInfo;     /* pointer to an M-Motion file    */
                                    /* status structure that we will */
                                    /* use for this file instance    */
    MMIMAGEHEADER     MMImgHdr;
    ULONG             ulRequiredFileLength;
    ULONG             ulActualFileLength;
    ULONG             ulWidth;
    ULONG             ulHeight;
    PBYTE             lpYUVBuf;
    ULONG             ulRowCount;
    ULONG             ulRGBBytesPerLine;
    ULONG             ulYUVBytesPerLine;
    LONG              rc;
    HMMIO             hmmioSS;
    PBYTE             lpRGBBufPtr;
    FOURCC            fccStorageSystem;    /* SS I/O Proc FOURCC       */
    MMIOINFO          mmioinfoSS;          /* I/O info block for SS ref */
    PSZ pszFileName = (CHAR *) lParam1;    /* get the filename from    */
                                           /* parameter                */
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Check for valid MMIOINFO block.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    if (!pmmioinfo)
        return (MMIO_ERROR);
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * If flags show read and write then send back an error. We
     * only support reading or writing but not both at the same
     * time on the same file.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    if ((pmmioinfo->ulFlags & MMIO_READWRITE) &&
        ((pmmioinfo->ulTranslate & MMIO_TRANSLATEDATA) ||
         (pmmioinfo->ulTranslate & MMIO_TRANSLATEHEADER)))
        {
```

```
     return (MMIO_ERROR);
     }
  /*************************
   *Determine the storage system/child IOProc that actually
   *obtains the data for us. The M-Motion data may be contained
   *in a memory (RAM) file, as a component in a database or
   *library (a Compound file), or as a stand-alone disk file.
   *
   *While the application uses this M-Motion IOProc to obtain
   *untranslated (M-Motion) or translated (bit map) data,
   *the IOProc must obtain it's data from something that
   *reads and writes to a storage media. The exact storage
   *media is immaterial-so long as the read and write
   *operations generate data that LOOKS like it is part
   *of a standard file.
   *************************/
  if (!pmmioinfo->fccChildIOProc)
     {
     /* Need to determine SS if create from pmmioinfo and filename. */
     if (pmmioinfo->ulFlags & MMIO_CREATE)
        {
        if (mmioDetermineSSIOProc( pszFileName,
                                   pmmioinfo,
                                   &fccStorageSystem,
                                   NULL ))
           {
           fccStorageSystem = FOURCC_DOS;
           }
        }
     else
        {
        if (mmioIdentifyStorageSystem( pszFileName,
                                       pmmioinfo,
                                       &fccStorageSystem ))
           {
           return (MMIO_ERROR);
           }
        }
     if (!fccStorageSystem)
        {
        return (MMIO_ERROR);
```

```
      }
    else
      {
      pmmioinfo->fccChildIOProc = fccStorageSystem;
      }
    }/* end storage system identification block */
   /*******************************************************
    * Direct the open to the specific storage system necessary
    *******************************************************/
    memset (&mmioinfoSS, '\0', sizeof (MMIOINFO));
    memmove (&mmioinfoSS, pmmioinfo, sizeof (MMIOINFO));
    mmioinfoSS.pIOProc = NULL;
    mmioinfoSS.fccIOProc = pmmioinfo->fccChildIOProc;
    mmioinfoSS.ulFlags |= MMIO_NOIDENTIFY;
   /*******************************************************
    * Try to open the file. Add the NO IDENTIFY flag to
    *     ENSURE THAT WE DON'T LOOP RECURSIVELY!!!
    *******************************************************/
    hmmioSS = mmioOpen (pszFileName,
                        &mmioinfoSS,
                        mmioinfoSS.ulFlags);
   /*******************************************************
    * Check if a DELETE was requested-mmioOpen returns a 1,
    *     so we much check this separately
    *******************************************************/
    if (pmmioinfo->ulFlags & MMIO_DELETE)
      {
     /* was the delete successful?          */
      if (!hmmioSS)
        {
        pmmioinfo->ulErrorRet = MMIOERR_DELETE_FAILED;
        return (MMIO_ERROR);
        }
      else
        {
        return (MMIO_SUCCESS);
        }
      }
   /*******************************************************
    * Check the return code from the open call for an error.
```

```
 *     If not delete, then the open should have worked.
 ********************************/
if (!hmmioSS)
    return (MMIO_ERROR);
/********************************
 * Allocate memory for one M-Motion FileStatus structures
 ********************************/
DosAllocMem ((PPVOID) &pVidInfo,
                    sizeof (MMFILESTATUS),
                    fALLOC);
/********************************
 * Ensure the allocate was successful.If not, then
 *     close the file and return open as unsucessful...
 ********************************/
if (!pVidInfo)
   {
   mmioClose (hmmioSS, 0);
   return (MMIO_ERROR);
   }
pVidInfo->hmmioSS = hmmioSS;
/********************************
 * Store pointer to our MMFILESTATUS structure in
 * pExtraInfoStruct field that is provided for our use.
 ********************************/
pmmioinfo->pExtraInfoStruct = (PVOID)pVidInfo;
/********************************
 * Set the fields of the FileStatus structure that the
 * IOProc is responsible for.
 ********************************/
InitFileStruct (pVidInfo);
/********************************
 * If this is a read, we need to check that is a M-Motion
 *     file and perhaps get the data.
 ********************************/
if (pmmioinfo->ulFlags & MMIO_READ)
   {
   /********************************
    * First we must get some basic information from the file
    * Read in data to fill up the MMOTIONHEADER structure.
    *
    * If the read is unsuccessful, this is not a M-Motion file
```

```
   * and we should return a failure on the open
   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  if (sizeof (MMOTIONHEADER) ! =
                    mmioRead (pVidInfo- > hmmioSS,
                              (PVOID) &pVidInfo- > mmotHeader,
                              (ULONG) sizeof (MMOTIONHEADER)))
    {
    mmioClose (pVidInfo- > hmmioSS, 0);
    DosFreeMem ((PVOID) pVidInfo);
    return (MMIO_ERROR);
    }
  / * Ensure this IS an M-Motion file header before we continue */
  if (strcmp (pVidInfo- > mmotHeader.mmID, "YUV12C"))
     {
    mmioClose (pVidInfo- > hmmioSS, 0);
    DosFreeMem ((PVOID) pVidInfo);
    return (MMIO_ERROR);
    }
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Set up width and height of image.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulWidth = (ULONG)pVidInfo- > mmotHeader.mmXlen;
ulHeight = (ULONG)pVidInfo- > mmotHeader.mmYlen;
/ * Calculate what the length of the file SHOULD be based on the */
/ *     header contents                                           */
ulRequiredFileLength = (((ulWidth > > 2) * 6) * ulHeight) +
                                         sizeof (MMOTIONHEADER);
/ * Query what the ACTUAL length of the file is,                   */
/ *     then move back to Just after the header.
ulActualFileLength = (ULONG) mmioSeek (pVidInfo- > hmmioSS,
                                      0, SEEK_END);
mmioSeek (pVidInfo- > hmmioSS, sizeof (MMOTIONHEADER), SEEK_SET);
/ * If these don't match, then it isn't a VALID M-Motion file      */
/ *     -regardless of what the header says.                       */
if (ulRequiredFileLength ! = ulActualFileLength)
   {
  mmioClose (pVidInfo- > hmmioSS, 0);
  DosFreeMem ((PVOID) pVidInfo);
  return (MMIO_ERROR);
  }
```

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * If the app intends to read in translation mode, we must
  * allocate and set-up the buffer that will contain the RGB data .
  *
  * We must also read in the data to insure that the first
  * read, seek, or get-header operation will have data
  * to use .This is ONLY NECESSARY FOR TRANSLATED MODE
  * operations, since we must process reads/ writes pretending
  * the image is stored from the bottom-up .
  *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Fill out the MMIMAGEHEADER structure .
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
MMImgHdr .ulHeaderLength = sizeof (MMIMAGEHEADER);
MMImgHdr .ulContentType  = MMIO_IMAGE_PHOTO;
MMImgHdr .ulMediaType    = MMIO_MEDIATYPE_IMAGE;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cbFix =
            sizeof (BITMAPINFOHEADER2);
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cx              = ulWidth;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cy              = ulHeight;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cPlanes         = 1;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cBitCount       = 24;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .ulCompression   =
    BCA_UNCOMP;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cbImage         =
    ulWidth * ulHeight * 3;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cxResolution    = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cyResolution    = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cclrUsed        = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cclrImportant   = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .usUnits         = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .usReserved      = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .usRecording     =
    BRA_BOTTOMUP;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .usRendering     =
    BRH_NOTHALFTONED;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cSize1          = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .cSize2          = 0L;
MMImgHdr .mmXDIBHeader .BMPInfoHeader2 .ulColorEncoding = 0L;
```

```
MMImgHdr.mmXDIBHeader.BMPInfoHeader2.ulIdentifier     = 0L;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Copy the image header into private area for later use.
  * This will be returned on a mmioGetHeader () call
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
pVidInfo->mmImgHdr = MMImgHdr;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * YUV Bytes/Line are = 1 1/2 times the number of pels
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulYUVBytesPerLine = ulWidth + (ulWidth >> 1);
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * RGB Bytes/Line are  = 2 * YUV bytes/line
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulRGBBytesPerLine = (ulYUVBytesPerLine << 1);
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Determine total bytes in image
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
pVidInfo->ulRGBTotalBytes = ulWidth * ulHeight * 3;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * M-Motion images are always on 4-pel boundaries, which also
  * makes them on 4-byte/LONG boundaries, which is used for
  * bit maps. Therefore, there are no extra pad bytes necessary.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
pVidInfo->ulImgPaddedBytesPerLine = ulWidth * 3;
pVidInfo->ulImgTotalBytes = pVidInfo->ulRGBTotalBytes;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * For translated data READ mode, we must allocate a buffer,
  * get the YUV data from the file, and load the RGB buffer.
  * Place format-specific code here to load the image into the
  * buffer. The code below is M-Motion format specific.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
if (pmmioinfo->ulTranslate & MMIO_TRANSLATEDATA)
   {
   / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Get space for full image buffer.
     * This will be retained until the file is closed.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    if (DosAllocMem ((PPVOID) &(pVidInfo->lpRGBBuf),
                     pVidInfo->ulRGBTotalBytes,
                     fALLOC))
```

```
    {
    mmioClose (pVidInfo->hmmioSS, 0);
    DosFreeMem ((PVOID) pVidInfo);
    return (MMIO_ERROR);
    }
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Get temporary space for one line YUV buffer.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
if (DosAllocMem ((PPVOID) &lpYUVBuf,
                ulYUVBytesPerLine,
                fALLOC))
   {
   mmioClose (pVidInfo->hmmioSS, 0);
   DosFreeMem ((PVOID) pVidInfo);
   return (MMIO_ERROR);
   }
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Initialize the beginning buffer position.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
lpRGBBufPtr = pVidInfo->lpRGBBuf;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Read in YUV data one line at a time, converting
  * from YUV to RGB, then placing in the image buffer.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
for (ulRowCount = 0;
     ulRowCount < ulHeight;
     ulRowCount++)
    {
    / * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * Read in one line.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    rc = mmioRead (pVidInfo->hmmioSS,
                  (PVOID) lpYUVBuf,
                  (ULONG) ulYUVBytesPerLine);
    / * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * Convert one line at a time.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    ConvertOneLineYUVtoRGB (lpYUVBuf,
                            lpRGBBufPtr,
```

```
                        ulYUVBytesPerLine);
        /****************************
         * Make sure buffer ptr is correct for the next convert.
         ****************************/
        lpRGBBufPtr += (LONG) ulRGBBytesPerLine;
        }   /* end of FOR loop to read YUV data */
      DosFreeMem (lpYUVBuf);
      /*****************************
       * This changes from M-Motion's top-down form to OS/2
       * PM'S bottom-up bit map form.
       *****************************/
      ImgBufferFlip (pVidInfo->lpRGBBuf,
                     pVidInfo->ulImgPaddedBytesPerLine,
                     ulHeight);
      /*****************************
       * RGB buffer now full, set position pointers to the
       * beginning of the buffer.
       *****************************/
      pVidInfo->lImgBytePos = 0;
      }   /* end IF TRANSLATED block */
    }/* end IF READ block */
  return (MMIO_SUCCESS);
  }/* end case of MMIOM_OPEN */
```

图 4-7 MMIOM_OPEN 情况(MMOTPROC.C)

4.4.2 MMIOM_READ 和 MMIOM_WRITE

MMIOM_READ 消息要求从打开的文件读字节;MMIOM_WRITE 消息要求将字节写至一个打开的文件。不同的 IOProc 处理这些消息不同,取决于文件数据的要求。因为文件可能会用到缓冲 I/O, mmioRead 及 mmioWrite 具有 lBufOffset 和 lDiskOffset 字段。IOProc 不能修改这些字段。如果 IOProc 需要这些字段,则用 aulInfo 数组来获得。另外,pExtraInfoStruct 可用于所有 IOProc 要求的用户定义结构。IOProc 样例在该字段存储了头,以表示这种功能。如果 IOProc 是文件格式的,则须用 mmioRead 或 mmioWrite 调用存储系统 IOProc,采用打开过程中产生的内部句柄。存储系统 IOProc 只须简单地调用 DosRead 或 DosWrite。

要为翻译模式安装文件格式 IOProc,并提供对 MMIO_TRANSLATEDATA 标志的支持,则要求更多的 MMIOM_READ 和 MMIOM_WRITE 消息处理的代码。读处理期间,当数据已经从文件读至其原编码格式的私有缓存区后,数据必须从其原编码模式翻译

成媒体类型的标准显示格式编码模式。翻译的数据则显示在应用程序的读缓冲区。同样地,对于读处理,数据从应用程序接收来,是标准显示格式的,在读入文件前必须翻译成其原编码模式。

图4-8 **显示了一个** M-Motion IOProc **是如何支持** MMIOM_READ **和** MMIOM_WRITE **消息的例子。**

```
case MMIOM_READ:
    {
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Declare Local Variables
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    PMMFILESTATUS    pVidInfo;
    LONG             rc;
    LONG             lBytesToRead;
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Check for valid MMIOINFO block.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    if (!pmmioinfo)
        return (MMIO_ERROR);
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Set up our working file status variable.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Is Translate Data off?
     * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    if (!(pmmioinfo->ulTranslate & MMIO_TRANSLATEDATA))
        {
        /* * * * * * * * * * * * * * * * * * * * * * * * * * *
         * Since no translation, provide exact number of bytes req.
         * * * * * * * * * * * * * * * * * * * * * * * * * * */
        if (!lParam1)
            return (MMIO_ERROR);

        rc = mmioRead (pVidInfo->hmmioSS,
                      (PVOID) lParam1,
                      (ULONG) lParam2);

        return (rc);
        }
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
 * Otherwise, Translate Data is on ...
 ********************************/
/********************************
 * Ensure we do NOT write more data out than is remaining
 *    in the buffer.The length of read was requested in
 *    image bytes, so confirm that there are that many of
 *    virtual bytes remaining.
 ********************************/
if ((ULONG) (pVidInfo->lImgBytePos + lParam2) >
     pVidInfo->ulImgTotalBytes
   lBytesToRead =
       pVidInfo->ulImgTotalBytes-pVidInfo->lImgBytePos;
else
   lBytesToRead = (ULONG)lParam2;
/********************************
 * Perform this block on ALL reads.The image data should
 * be in the RGB buffer at this point, and can be handed
 * to the application.
 *
 * Conveniently, the virtual image position is the same
 *    as the RGB buffer position, since both are 24 bit-RGB
 ********************************/
memcpy ((PVOID)lParam1,
        &(pVidInfo->lpRGBBuf[pVidInfo->lImgBytePos]),
        lBytesToRead);
/********************************
 * Move RGB buffer pointer forward by number of bytes read.
 * The Img buffer pos is identical since both are 24 bits.
 ********************************/
pVidInfo->lImgBytePos += lBytesToRead;

return (lBytesToRead);
} /* end case of MMIOM_READ */

case MMIOM_WRITE;
 {
 /********************************
  * Declare Local Variables.
  ********************************/
 PMMFILESTATUS      pVidInfo;
 USHORT             usBitCount;
```

```
LONG                  lBytesWritten;
ULONG                 ulImgBytesToWrite;
/*******************************
 * Check for valid MMIOINFO block.
 *******************************/
if (!pmmioinfo)
   return (MMIO_ERROR);
/*******************************
 * Set up our working variable MMFILESTATUS.
 *******************************/
pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;
/*******************************
 * See if a SetHeader has been done on this file.
 *******************************/
if ((!pVidInfo) || (!pVidInfo->bSetHeader))
   {
   return (MMIO_ERROR);
   }
if (!(pmmioinfo->ulTranslate & MMIO_TRANSLATEDATA))
   {
   /****************************
    * Translation is off, take amount of bytes sent and
    * write to the file.
    ****************************
    * Ensure that there is a data buffer to write from.
    ****************************/
   if (!lParam1)
      return (MMIO_ERROR);
   lBytesWritten = mmioWrite (pVidInfo->hmmioSS,
                              (PVOID) lParam1,
                              (ULONG) lParam2);
   return (lBytesWritten);
   }
/*******************************
 * Translation is on.
 *******************************
 * Set up local variables.
 *******************************/
usBitCount =
```

```
        pVidInfo->mmImgHdr.mmXDIBHeader.BMPInfoHeader2.cBitCount;

    /********************************
     * Ensure we do not attempt to write past the end of the
     *    buffer...
     ********************************/
    if ((ULONG) (pVidInfo->lImgBytePos + lParam2) >
       pVidInfo->ulImgTotalBytes)
       ulImgBytesToWrite =
           pVidInfo->ulImgTotalBytes-pVidInfo->lImgBytePos;
    else
       ulImgBytesToWrite = (ULONG)lParam2;

    /********************************
     * Write the data into the image buffer. It will be converted to
     *   RGB, then YUV when the file is closed. This allows the
     *   application to seek to arbitrary positions within the
     *   image in terms of the bits/pel, etc they are writing.
     ********************************/
    memcpy (&(pVidInfo->lpImgBuf[pVidInfo->lImgBytePos]),
            (PVOID)lParam1,
            ulImgBytesToWrite);

    /* Update current position in the image buffer */
    pVidInfo->lImgBytePos += ulImgBytesToWrite;

    return (ulImgBytesToWrite);
    }  /* end case of MMIOM_WRITE */

/*
 * If the IOProc has a child IOProc, then pass the message
 *   on to the Child, otherwise return, Unsupported Message
 */
default:
    {
     /*
      * Declare Local Variables.
      */
     PMMFILESTATUS         pVidInfo;
     LONG                  lRc;

     /********************************
      * Check for valid MMIOINFO block.
      ********************************/
```

```
    if ( ! pmmioinfo)
       return (MMIO_ERROR);
   /*****************************
    * Set up our working variable MMFILESTATUS.
    *****************************/
    pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;
    if (pVidInfo != NULL && pVidInfo->hmmioSS)
      {
       lRC = mmioSendMessage (pVidInfo->hmmioSS,
                              usMsg,
                              lParam1,
                              lParam2);
       if ( ! lRC)
          pmmioinfo->ulErrorRet = mmioGetLastError (pVidInfo->hmmioSS);
       return (lRC);
      }
    else
      {
       if (pmmioinfo != NULL)
          pmmioinfo->ulErrorRet = MMIOERR_UNSUPPORTED_MESSAGE;
       return (MMIOERR_UNSUPPORTED_MESSAGE);
      }
    }  /* end case of Default */
   } /* end SWITCH statement for MMIO messages */
   return (0);
   }    /* end of window procedure */
```

图 4-8 MMIOM_READ 和 MMIOM_WRITE 情况(MMOTPROC.C)

4.4.3 MMIOM_SEEK

不同的 IOProc 对该消息的处理不同。因为有不同文件格式的要求,所以必须确定是否支持该消息,以及支持的话,如何安装三种不同类型的查找。文件格式 IOProc 可以使用某些运算来决定查找的距离及其单位。文件格式 IOProc 必须用 mmioSeek 来调用存储系统 IOProc。存储系统 IOProc 可以调用 DOSSetFilePtr 函数来设置文件的位置。对于组合文件元素,装订检验由 BND 文件元素的 mmioSeek 函数处理。具有该特性的任何其它组合文件 IOProc 必须在其自己的代码中安装这种检验。

图 4-9 显示了一个 M-Motion IOProc 如何支持 MMIOM_SEEK 消息的例子。

```
case MMIOM_SEEK:
    {
    /*******************************
     * Set up locals.
     *******************************/
    PMMFILESTATUS     pVidInfo;
    LONG              lNewFilePosition;
    LONG              lPosDesired;
    SHORT             sSeekMode;
    /*******************************
     * Check to make sure MMIOINFO block is valid.
     *******************************/
    if ( ! pmmioinfo)
        return (MMIO_ERROR);
    /*******************************
     * Set up our working file status variable.
     *******************************/
    pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;

    lPosDesired = lParam1;
    sSeekMode = (SHORT) lParam2;
    /*******************************
     * Is Translate Data on?
     *******************************/
    if (pmmioinfo->ulTranslate & MMIO_TRANSLATEDATA)
        {
        /*******************************
         * Attempt to move the Image buffer pointer to the
         * desired location.App sends SEEK requests in
         * positions relative to the image planes & bits/pel
         * We must also convert this to RGB positions
         *******************************/
        switch (sSeekMode)
            {
            case SEEK_SET;
                {
                lNewFilePosition = lPosDesired;
                break;
                }
            case SEEK_CUR;
                {
```

```
            lNewFilePosition = pVidInfo->lImgBytePos + lPosDesired;
            break;
            }
         case SEEK_END;
            {
            lNewFilePosition =
                         pVidInfo->ulImgTotalBytes + = lPosDesired;
            break;
            }
         default:
            return (MMIO_ERROR);
         }
    /******************************
     * Make sure seek did not go before start of file.
     * If so, then don't change anything, just return an error
     ******************************/
    if (lNewFilePosition < 0)
       {
       return (MMIO_ERROR);
       }
    /******************************
     * Make sure seek did not go past the end of file.
     ******************************/
    if (lNewFilePosition > (LONG) pVidInfo->ulImgTotalBytes)
        lNewFilePosition = pVidInfo->ulImgTotalBytes;
    pVidInfo->lImgBytePos = lNewFilePosition;
    return (pVidInfo->lImgBytePos);
    } /* end IF DATA TRANSLATED */
/*******************************
 * Translate Data is OFF...
 *******************************
 * if this is a seek from the beginning of the file,
 *     we must account for and pass the header
 *******************************/
if (lParam2 = = SEEK_SET)
   lPosDesired + = MMOTION_HEADER_SIZE;
lNewFilePosition = mmioSeek (pVidInfo->hmmioSS,
                             lPosDesired,
                             sSeekMode);
```

```
    /**********************
     * Ensure we did not move to within the header
     **********************/
    if ((lNewFilePosition != MMIO_ERROR) &&
       (lNewFilePosition < MMOTION_HEADER_SIZE))
      {
      lNewFilePosition = mmioSeek (pVidInfo->hmmioSS,
                                   (LONG) MMOTION_HEADER_SIZE,
                                   SEEK_SET);
      }

    /*****************************
     * Return new position. Always remove the length of the
     *     header from the this position value
     *****************************/
    if (lNewFilePosition != MMIO_ERROR)
      lNewFilePosition -= MMOTION_HEADER_SIZE;

    return (1NewFilePosition);
    } /* end case of MMIOM_SEEK */
```

图 4-9 MMIOM_SEEK 情况(MMOTPROC.C)

4.4.4 MMIOM_CLOSE

该消息必须由文件格式 IOProc 安装,用于正确关闭文件。注意 mmioClose 函数调用 mmioFlush 函数来清文件 I/O 缓冲区(所以不必采用 IOProc 消息句柄)。

文件格式 IOProc 调用带内部 HMMIO 句柄的 mmioClose 函数来关闭文件,除非它能能够识别 mmioClose 上的 MMIO_FHOPEN 标志。该标志允许关闭的同时文件句柄保持打开。如果文件是组合文件的一个元素,也许需要更新组合文件头以反映对文件所做的改变。

图 4-10 显示了一个 M-Motion IOProc 如何支持 MMIOM_CLOSE 消息的例子。

```
Case MMIOM_CLOSE:
    {
   /******************************
     * Declare local variables.
     ******************************/
    PMMFILESTATUS    pVidInfo;        /* MMotionIOProc instance data */
    ULONG            ulHeight;        /* Image height                */
```

```
USHORT          usBitCount;                                              */
/* Image width,  including overflow in 1bpp & 4bpp                       */
ULONG           ulImgPelWidth;
PBYTE           lpYUVLine;         /* one line of packed YUV             */
LONG            lYUVBytesPerLine;
ULONG           ulMaxPelWidth;     /* # pels on 4-pel boundaries         */
/* # pels on a YUV line in the output file                               */
ULONG           ulYUVPelWidth;
ULONG           ulRGBMaxBytesPerLine;/* # bytes on 4-pel bounds          */
PBYTE           lpRGBLine;         /* One line of 24bit RGB              */
PBYTE           lpImgBufPtr;       /* Current loc in RGB image buf       */
LONG            lRGBBytesPerLine;  /* # bytes on a line in image         */
ULONG           ulRowCount;        /* loop counter                       */
LONG            lBytesWritten;     /* # bytes output on a write          */
LONG            lRetCode;
USHORT          rc;
/*******************************
 * Check for valid MMIOINFO block.
 *******************************/
if (!pmmioinfo)
   return (MMIO_ERROR);
/*******************************
 * Set up our working file status variable.
 *******************************/
pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;
/*******************************
 * Assume success for the moment....
 *******************************/
lRetCode = MMIO_SUCCESS;
/*******************************
 * see if we are in Write mode and have a buffer to write out.
 *    We have no image buffer in UNTRANSLATED mode.
 *******************************/
if ((pmmioinfo->ulFlags & MMIO_WRITE) && (pVidInfo->lpImgBuf))
   {
   /*****************************
    * The buffer shoule be in palettized or 24-bit RGB
    * We must convert it to YUV to be written to the file.
    *****************************
```

```
  * The buffer should be complete . If not, then we
  * should still close, but can flag an error to the
  * user that the data may be corrupted . The only way
  * we can estimate if this is true is to check the final
  * position . If not at the end ...
  * * * * * * * * * * * * * * * * * * * * * * * * * * * */
if (pVidInfo- > lImgBytePos ! =
                     (LONG) pVidInfo- > ulImgTotalBytes)
   {
   lRetCode = MMIO_WARNING;
   }
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Set up width and height of image in the buffer .
 * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulHeight = pVidInfo- > mmImgHdr.mmXDIBHeader.BMPInfoHeader2.cy;
usBitCount =
   pVidInfo- > mmImgHdr.mmXDIBHeader.BMPInfoHeader2.cBitCount;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Get the line width in YUV pels and packed bytes .
 * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulYUVPelWidth = pVidInfo- > mmotHeader.mmXlen;
lYUVBytesPerLine = (LONG) (ulYUVPelWidth * 3) > > 1;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * This changes from OS/ 2 PM bottom-up bitmap form
 *     to M-Motion's top-down form . Flip all pad, boundary
 *     bytes as well
 * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ImgBufferFlip ((PBYTE)pVidInfo- > lpImgBuf,
                  pVidInfo- > ulImgPaddedBytesperLine,
                  ulHeight);
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * Determine number of POSSIBLE pels on a line, tho some
 *     may be overflow in 1bpp and 4bpp modes .
 *
 * From that, we can calc the number of RGB pels we will
 *     create to represent this line .
 * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulImgPelWidth = (pVidInfo- > ulImgpelBytesPerLine < < 3) /
                      usBitCount;
lRGBBytesperLine = ulImgPelWidth * 3;
```

```
/*****************************
 * Ensure the width is on a 4-pel boundary, necessary for
 *      M-Motion. We will buffer with black
 *     *** THIS IS ONLY NECESSARY FOR M-MOTION IMAGES
 *****************************/
if (ulImgPelWidth % 4)
    ulMaxPelWidth = (((ulImgPelWidth >> 2) + 1) << 2);
else
    ulMaxpelWidth = ulImgPelWidth;

/* # RGB bytes/line = # pels * 3 bytes/pel */
ulRGBMaxBytesPerLine = ulMaxPelWidth * 3;

/*****************************
 * Create a buffer for one line of RGB data, accounting for
 *     the 4-pel boundary required. Extra bytes won't be used.
 *****************************/
if (DosAllocMem ((PPVOID) &lpRGBLine,
                 ulRGBMaxBytesPerLine,
                 fALLOC))
    return (MMIO_ERROR);

/*****************************
 * Create a buffer for one line of YUV data.
 *****************************/
if (DosAllocMem ((PPVOIO) &lpYUVLine,
                 lYUVBytesPerLine,
                 fALLOC))
    return (MMIO_ERROR);

/*****************************
 * Zero out RGB buffer to cover for any extra black pels
 * needed at the end.
 *****************************/
memset (lpRGBLine, 0, ulRGBMaxBytesPerLine);

/*****************************
 * Initialize start position of RGB buffer.
 *****************************/
lpImgBufPtr = pVidInfo->lpImgBuf;

/*****************************
 * process Image Buffer - Save to file
 * Place "your" processing code here, if full image
 *      buffering is performed.
```

```
 * For M-Motion:
 *     LOOP
 *         1 . Convert and copy a line of lbpp, 4bpp, 8 bpp or
 *              24bpp data into a temporary 24 bpp RGB line .
 *              This line may contain overflow garbage
 *              pels from lbpp and 4bpp modes (where
 *              width does not fall on even byte boundaries .)
 *         2 . Convert the temporary RGB line contents into a
 *              YUV line . ONLY that data necessary converted .
 *              Overflow from bitmap data ignored .
 *         3 . Write the YUV line to the file
 * * * * * * * * * * * * * * * * * * * * * * * * * * * */
for (ulRowCount = 0;
     ulRowCount < ulHeight;
     ulRowCount + + )
    {
   / * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Convert 1 line of Image data into RGB data
     * * * * * * * * * * * * * * * * * * * * * * * * * */
    switch (usBitCount)
       {
       case 1 :
           {
           / * Convert lbpp padded image buffer into 24-bit */
           / *    RGB line buffer, w/ pads                  */
           Convert1 BitTo24Bit (
                   (PBYTE) lpImgBufPtr,
                   (PRGB) lpRGBLine,
                   (PRGB) &(pVidInfo- > rgbPalette),
                   pVidInfo- > ulImgPelBytesPerLine);
           break;
           }
       case 4 :
           {
           / * Convert data from app buffer into 24-bit and */
           / *    copy into image buffer                    */
           Convert4 BitTo24Bit (
                   (PBYTE) lpImgBufPtr,
                   (PRGB) lpRGBLine,
                   (PRGB) &(pVidInfo- > rgbPalette),
                   pVidInfo- > ulImgPelBytesPerLine);
```

```
        break;
        }
    case 8:
         {
        /* Convert data from app buffer into 24-bit and */
        /*     copy into image buffer                    */
         Convert8BitTo24Bit (
                 (PBYTE)lpImgBufPtr,
                 (PRGB) lpRGBLine,
                 (PRGB) &(pVidInfo->rgbPalette),
                 pVidInfo->ulImgPelBytesPerLine);
         break;
         }
    case 24:
         {
        /* Copy raw RGB data from the image buffer into */
        /*     the temporary                              */
        /*    RGB line. Only copy those pels necessary. */
        /*     No conversion required                     */
         memcpy((PVOID)lpRGBLine,
                       (PVOID) lpImgBufPtr,
                       ulYUVPelWidth * 3);
         break;
         }
    }  /* end of Switch for Bit Conversion block */
/***************************
 * convert one line at a time from RGB to YUV.
 ***************************/
ConvertOneLineRGBtoYUV(lpRGBLine,
                       lpYUVLine,
                       ulYUVPelWidth);

/***************************
 * Write out line of YUV data to the file.
 ***************************/
lBytesWritten = mmioWrite(pVidInfo->hmmioSS,
                          (PVOID) lpYUVLine,
                          lYUVBytesPerLine);
/* Check if error or EOF */
```

```
        if (lBytesWritten != lYUVBytesPerLine)
          {
          lRetCode = lBytesWritten;
          break;
          }
       /*****************************
        * Make sure bitmap image buffer pointer is correct
        *    for next line to be converted.Move forward ALL
        *    the bytes in the bitmap line, including overflow
        *    and pad bytes.
        *****************************/
        lpImgBufPtr += pVidInfo->ulImgPaddedBytesPerLine;
        }
    /*****************************
     * Free temp buffers.
     *****************************/
     if (lpRGBLine)
       {
       DosFreeMem ((PVOID) lpRGBLine);
       }
     if (lpYUVLine)
       {
       DosFreeMem ((PVOID) lpYUVLine);
       }
     } /* end IF WRITE & IMAGE BUFFER block */
    /*******************************
     * Free the RGB buffer, if it exists, that was created
     * for the translated READ operations.
     *******************************/
     if (pVidInfo->lpRGBBuf)
       {
       DosFreeMem ((PVOID) pVidInfo->lpRGBBuf);
       }
    /*******************************
     * Free the IMG buffer, if it exists, that was created
     * for the translated WRITE operations.
     *******************************/
     if (pVidInfo->lpImgBuf)
       {
```

```
            DosFreeMem ((PVOID) pVidInfo->lpImgBuf);
          }
        /*******************************
         * Close the file with mmioClose.
         *******************************/
        rc = mmioClose (pVidInfo->hmmioSS, 0);

        DosFreeMem ((PVOID) pVidInfo);

        if (rc != MMIO_SUCCESS)
          return (rc);

        return (lRetCode);
        }  /* end case of MMIOM_CLOSE */
```

图 4-10　MMIOM_CLOSE 情况(MMOTPROC.C)

4.4.5　MMIOM_IDENTIFYFILE

IOProc决定如何处理该条消息。所有的 IOProc 都必须支持这条消息,因为 mmioOpen 函数在试图自动识别一个文件时会发送 MMIOM_IDENTIFYFILE。

对于文件格式 IOProc,需要读取头数据,检验是否与 IOProc 所要的匹配。lParm2 字段包含了用于 mmioRead 函数的句柄。图 4-11 中,头是一个 M-Motion 文件,与 IOProc 中定义的字符串去比较,如果比较结果正确,消息返回 TRUE,否则返回 FALSE。

```
case MMIOM_IDENTIFYFILE:
    {
    /*******************************
     * Declare local variables.
     *******************************/
    MMOTIONHEADER    mmotHeader:    /* M-Motion structure variable */
    HMMIO            hmmioTemp;     /* MMIO File Handle            */
    ULONG            ulWidth;
    ULONG            ulHeight;
    ULONG            ulRequiredFileLength;
    ULONG            ulActualFileLength;
    BOOL              fValidMMotionFile = FALSE;

    ULONG            ulTempFlags = MMIO_READ | MMIO_DENYWRITE |
MMIO_NOIDENTIFY;
                                    /* Flags used for temp open    */
                                    /* and close                   */
      /*******************************
```

```
 * We need either a file name (lParam1) or file handle (lParam2)
 ******************************/
if (!lParam1 && !lParam2)
   return (MMIO_ERROR);

/* Copy the file handle, assuming one was provided ... */
hmmioTemp = (HMMIO)lParam2;

/*******************************
 * If no handle, then open the file using the string name
 ******************************/
if (!hmmioTemp)
  {
  if (!(hmmioTemp = mmioOpen ((PSZ) lParam1,
                              NULL,
                              ulTempFlags)))
      {
      return (MMIO_ERROR);
      }
  }

/*******************************
 * Read in enough bytes to check out file.
 ******************************/
if (sizeof (MMOTIONHEADER) !=
                mmioRead (hmmioTemp,
                          (PVOID) &mmotHeader,
                          (ULONG) sizeof (MMOTIONHEADER)))
   {
   /*****************************
    * Fail so close file and then return.
    *****************************/
   if (!lParam2)/* Don't close handle if provided to us */
      mmioClose (hmmioTemp, 0);
   return (MMIO_ERROR);
   }

/*******************************
 * Close file before returning.
 ******************************/
if (!lParam2)/* Don't close handle if provided to us */
  mmioClose (hmmioTemp, 0);

/*******************************
```

```
  * Check validity of file and return result.
  *************************************************/
if (memcmp (mmotHeader.mmID, "YUV12C", 6) == 0)
   {
   ulWidth = mmotHeader.mmXlen;
   ulHeight = mmotHeader.mmYlen;

   /* Calculate what the length of the file SHOULD be based on the */
   /*   header contents                                             */
   ulRequiredFileLength = (((ulWidth >> 2) * 6) * ulHeight) +
                              sizeof (MMOTIONHEADER);

   /* Query what the ACTUAL length of the file is                   */
   ulActualFileLength = (ULONG) mmioSeek (hmmioTemp, 0, SEEK_END);

   /* If these don't match, then it isn't a VALID M-Motion file     */
   /*      - regardless of what the header says.                    */
   if (ulRequiredFileLength == ulActualFileLength)
      fValidMMotionFile = TRUE;
   else
      fValidMMotionFile = FALSE;
   } /* end header check block */
/**************************************************
  * Close file before returning.
  *************************************************/
if (!lParam2) /* Don't close handle if provided to us          */
   mmioClose (hmmioTemp, 0);

if (fValidMMotionFile)
   return (MMIO_SUCCESS);
else
   return (MMIO_ERROR);
}/* end case of MMIOM_IDENTIFYFILE */
```

图 4-11　MMIOM_IDENTIFYFILE 情况(MMOTPROC.C)

4.4.6　MMIOM_GETFORMATINFO

该消息要求 IOProc 格式的信息。MMIO 提供一组 MMFORMATINFO 结构,包括了有关当前所安装的 IOProc 支持的格式的描述性信息;例如,格式名、FOURCC 标识符以及相关的信息。如果 IOProc 中未定义该条消息或 IOProc 不能成功地处理这条消息,则 mmioGetFormatInfo 函数创建一个空白的 MMFORMATINFO 结构并将其放在内部列表中。建议对 ulStrucLen, fccIOProc, ulMediaType 和 ulFlags 字段的 IOProc 消息句柄码的

实际格式信息进行硬编码。另外，为考虑到 NLS **的需要，还应在源文件中存储其它信息**(ulCodePage, ulLanguage, lNameLength **以及** aulDefaultFormatExt **等等**)。

图 4-12 **显示了一个** M-Motion IOProc **如何支持** MMIOM_GETFORMATINFO **的例子**。

```
case MMIOM_GETFORMATINFO:
    {
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Declare local variables.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    PMMFORMATINFO        pmmformatinfo;
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Set pointer to MMFORMATINFO structure.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    pmmformatinfo = (PMMFORMATINFO) lParam1;
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     * Fill in the values for the MMFORMATINFO structure.
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    pmmformatinfo->ulStructLen    = sizeof (MMFORMATINFO);
    pmmformatinfo->fccIOProc      = FOURCC_MMOT;
    pmmformatinfo->ulIOProcType   = MMIO_IOPROC_FILEFORMAT;
    pmmformatinfo->ulMediaType    = MMIO_MEDIATYPE_IMAGE;
    pmmformatinfo->ulFlags        = MMIO_CANREADTRANSLATED
                                    MMIO_CANREADUNTRANSLATED
                                    MMIO_CANWRITETRANSLATED
                                    MMIO_CANWRITEUNTRANSLATED
                                    MMIO_CANREADWRITEUNTRANSLATED
                                    MMIO_CANSEEKTRANSLATED
                                    MMIO_CANSEEKUNTRANSLATED;
    strcpy ((PSZ) pmmformatinfo->szDefaultFormatExt, pszMotionExt);
    if (GetNLSData ( &pmmformatinfo->ulCodePage,
                     &pmmformatinfo->ulLanguage ))
      {
      return( -1L )
      }
    if (GetFormatStringLength ( FOURCC_MMOT,
                                &(pmmformatinfo->lNameLength) ))
      {
      return( -1L );
```

```
        }

        /*******************************
         * Return success back to the application.
         *******************************/
        return (MMIO_SUCCESS);
        }/* end case of MMIOM_GETFORMATINFO */
```

图 4-12 MMIOM_GETFORMATINFO 情况(MMOTPROC.C)

4.4.7 MMIOM_GETFORMATNAME

该消息要求由 IOProc 支持的描述性格式名字。建议为考虑 NLS 的需要在源文件中包括字符串。

图 4-13 显示了一个 M-Motion IOProc 如何支持 MMIOM_GETFORMATNAME 消息的例子。

```
case MMIOM_GETFORMATNAME:
    {
    LONG lBytesCopied;
   /*******************************
     * Copy the M-Motion format string into buffer supplied by
     * lParam1.Only put in the amount of my string up to the
     * allocated amount which is in lParam2.Leave enough room
     * for the NULL termination.
     *******************************/
    lBytesCopied = GetFormatString( FOURCC_MMOT,
                                    (char *)lParam1,
                                    lParam2 );

    return (lBytesCopied);
    }/* end case of MMIOM_GETFORMATNAME */
```

图 4-13 MMIOM_GETFORMATNAME 情况(MMOTPROC.C)

4.4.8 MMIOM_QUERYHEADERLENGTH

该消息要求 IOProc 返回由 mmioOpen 函数打开的当前文件或文件元素的头的大小。mmioQueryHeaderLength 函数发出一条 MMIOM_QUERYHEADERLENGTH 消息以确定 mmioGetHeader 要获得头数据所需的缓冲区大小。这一点很必要,因为头的长度各有不同。

要安装这条消息,可用 mmioSeek 存取当前文件位置,然后发出一个 mmioRead 调用,读取头的大小至缓冲区中。读操作可以不查找就完成,因为 mmioQueryHeaderLength 在调用时存取了当前的文件位置。该函数将搜索至文件头,然后搜索到 IOProc 调用时存取的文件位置。这里如果文件正在使用缓冲区 I/O,则使用 mmioRead 很重要,它可以使消息处理过程中所有的 MMIO 的内部数据字段均得到正确的维护。它还允许在该消息调用后,在文件的正确位置进行连续的文件读。

图 4-14 显示了一个 M-Motion IOProc 如何支持 MMIOM_QUERYHEADERLENGTH 消息的例子。

```
case MMIOM_QUERYHEADERLENGTH:
    {
   /******************************
     * If there is no MMIOINFO block then return an error.
     ******************************/
    if (!pmmioinfo)
        return (0);
   /******************************
     * If header is in translated mode then return the media
     * type specific structure size.
     ******************************/
    if (pmmioinfo->ulTranslate & MMIO_TRANSLATEHEADER)
        return (sizeof (MMIMAGEHEADER));
    else
        /******************************
         * Header is not in translated mode so return the size
         * of the M-Motion header.
         ******************************/
         return (sizeof (MMOTIONHEADER));
    break;
    }/* end case of MMIOM_QUERYHEADERLENGTH */
```

图 4-14　MMIOM_QUERYHEADERLENGTH 情况(MMOTPROC.C)

4.4.9　MMIOM_GETHEADER

该消息要求 IOProc 返回打开供 mmioOpen 读的当前文件或文件元素的指定的头信息;例如,媒体类型、媒体结构以及媒体结构的尺寸等。当用户在文件头中调用 mmioRead 读时,可使用头的大小,该数据由 lParam2 参数传递。

当翻译头为 TRUE 时,要求 IOProc 为这个媒体类型返回标准显示头结构。IOProc 必须将该文件的原始头数据转换进入这个结构。

图 4-15 显示了一个 M-Motion IOProc 如何支持 MMIOM_GETHEADER 消息的例

子。

```
case MMIOM_GETHEADER:
    {
    /*****************************
     * Declare local variables.
     *****************************/
    PMMFILESTATUS        pVidInfo;
    /*****************************
     * Check for valid MMIOINFO block.
     *****************************/
    if (!pmmioinfo)
       return (0);
    /*****************************
     * Set up our working file status variable.
     *****************************/
    pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;
    /*****************************
     * Getheader only valid in READ or READ/WRITE mode.
     * There is no header to get in WRITE mode.We
     * must aiso have a valid file handle to read from
     *****************************/
    if ((pmmioinfo->ulFlags & MMIO_WRITE) ||
       (!(pVidInfo->hmmioSS)))
      return (0);
    /*****************************
     * Check for Translation mode.
     *****************************/
    if (!(pmmioinfo->ulTranslate & MMIO_TRANSLATEHEADER))
       {
       /*****************************
        * Translation is off.
        *****************************/
       if (lParam2 < sizeof (MMOTIONHEADER))
          {
          pmmioinfo->ulErrorRet = MMIOERR_INVALID_BUFFER_LENGTH;
          return (0);
          }
       if (!lParam1)
```

```
            {
            pmmioinfo->ulErrorRet = MMIOERR_INVALID_STRUCTURE;
            return (0);
            }
        /***************************
          * Read in first 16 bytes to fill up M-Motion header.
          ***************************/
        memcpy ((PVOID) lParam1,
               (PVOID)&pVidInfo->mmotHeader,
               sizeof (MMOTIONHEADER));

        /* Indicate that the header has been set, which      */
        /* is meaningless in read mode, but allows the       */
        /* application to do writes in read/write mode       */
        pVidInfo->bSetHeader = TRUE;

        return (sizeof (MMOTIONHEADER));
        }   /* end IF NOT TRANSLATED block */
    /*********
      * TRANSLATION IS ON
      *********/
    if (lParam2 < sizeof (MMIMAGEHEADER))
        {
        pmmioinfo->ulErrorRet = MMIOERR_INVALID_BUFFER_LENGTH;

        return (0);
        }
    if (!lParam1)
        {
        pmmioinfo->ulErrorRet = MMIOERR_INVALID_STRUCTURE;
        return (0);
        }
    memcpy ((PVOID) lParam1,
           (PVOID)&pVidInfo->mmImgHdr,
           sizeof (MMIMAGEHEADER));
    return (sizeof (MMIMAGEHEADER));
    }/* end case of MMIOM_GETHEADER */
```

图 4-15 MMIOM_GETHEADER 情况(MMOTPROC.C)

4.4.10 MMIOM_SETHEADER

该消息要求 IOProc 设置打开供 mmioOpen 写的文件的指定头信息。这包括诸如图

象的分辨率、颜色以及音频的脉宽及抽样率等数据。

当翻译头为 TRUE 时，希望 IOProc 为那个媒体类型传送标准显示头。IOProc 应在写头入文件之前将数据从该结构转换进入其原始头结构。

图 4-16 显示了一个 M-Motion IOProc 如何支持 MMIOM_SETHEADER 消息的例子。

```
case MMIOM_SETHEADER:
    {
    /*****************************
     * Declare local variables.
     *****************************/
    PMMIMAGEHEADER      pMMImgHdr;
    PMMFILESTATUS       pVidInfo;
    USHORT              usNumColors;
    ULONG               ulImgBitsperLine;
    ULONG               ulImgBytesperLine;
    ULONG               ulBytesWritten;
    ULONG               ulWidth;
    ULONG               ul4PelWidth;
    ULONG               ulHeight;
    USHORT              usPlanes;
    USHORT              usBitCount;
    USHORT              usPadBytes;
    /*****************************
     * Check for valid MMIOINFO block.
     *****************************/
    if (!pmmioinfo)
       return (MMIO_ERROR);
    /*****************************
     * Set up our working variable MMFILESTATUS.
     *****************************/
    pVidInfo = (PMMFILESTATUS) pmmioinfo->pExtraInfoStruct;
    /*****************************
     * Only allow this function if we are in WRITE mode
     * And only if we have not already set the header
     *****************************/
    if ((!pmmioinfo->ulFlags & MMIO_WRITE)) ||
       (!(pVidInfo->hmmioSS)) ||
       (pVidInfo->bSetHeader))
       return (0);
```

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Make sure lParam1 is a valid pointer
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 if ( ! lParam1 )
    {
    pmmioinfo->ulErrorRet = MMIOERR_INVALID_STRUCTURE;
    return (0);
    }
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Header is not in translated mode.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 if ( ! (pmmioinfo->ulTranslate & MMIO_TRANSLATEHEADER))
    {
    / * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * Make sure lParam2 is correct size
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    if (lParam2 != MMOTION_HEADER_SIZE)
        {
        pmmioinfo->ulErrorRet = MMIOERR_INVALID_BUFFER_LENGTH;
        return (0);
        }
    / * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * Ensure that the header at least begins with "YUV12C"
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    if (strncmp ((char *)lParam1, "YUV12C", 6))
       {
       pmmioinfo->ulErrorRet = MMIOERR_INVALID_STRUCTURE;
       return (0);
       }
    / * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * Take 16 byte buffer (lParam1), write to file and.
      *     copy to internal structure.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    memcpy ((PVOID) &pVidInfo->mmotHeader,
            (PVOID)lParam1, (ULONG) MMOTION_HEADER_SIZE);
    ulBytesWritten = mmioWrite (pVidInfo->hmmioSS,
                                (PVOID) lParam1,
                                (ULONG) MMOTION_HEADER_SIZE);

    / * * * * * * * * * * * * * * * *
```

```
      * Check for an error on the write..
      * * * * * * * * * * * * * * * * */
    if (ulBytesWritten ! = MMOTION_HEADER_SIZE)
        return (0);  /* 0 indicates error */
    /* * * * * * * * * * * * * * * * *
      * Success...
      * * * * * * * * * * * * * * * * */
    pVidInfo->hSetHeader = TRUE;
    return (sizeof (MMOTIONHEADER));
    }  /* end IF NOT TRANSLATED block */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Header is translated.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Create local pointer media specific structure.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
pMMImgHdr = (PMMIMAGEHEADER) lParam1;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Check for validity of header contents supplied
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  *  -- Length must be that of the standard header
  *  -- NO Compression
  *     1 plane
  *     24, 8, 4 or 1 bpp
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
usBitCount = pMMImgHdr->mmXDIBHeader.BMPInfoHeader2.cBitCount;
if ((pMMImgHdr->mmXDIBHeader.BMPInfoHeader2.ulCompression ! =
     BCA_UNCOMP) ||
    (pMMImgHdr->mmXDIBHeader.BMPInfoHeader2.cPlanes ! = 1) ||
    ( !((usBitCount = = 24) || (usBitCount = = 8) ||
        (usBitCount = = 4) || (usBitCount = = 1)))
    )
     {
     pmmioinfo->ulErrorRet = MMIOERR_INVALID_STRUCTURE;
     return (0);
     }
if (lParam2 ! = sizeof (MMIMAGEHEADER))
   {
   pmmioinfo->ulErrorRet = MMIOERR_INVALID_BUFFER_LENGTH;
```

```
    return (0);
    }
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Complete MMOTIONHEADER.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
memcpy((PVOID)&pVidInfo->mmotHeader.mmID, "YUV12C", 6);
pVidInfo->mmotHeader.mmXorg = 0;
pVidInfo->mmotHeader.mmYorg = 0;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Ensure we will save stuff on 4-pel boundaries when
  * we actually convert to YUV and pack the bits.
  * We don't change what the user is actually going to
  * give us. The user thinks he is on 1-pel boundaries,
  * and that is how we buffer the RGB data.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulWidth  = pMMImgHdr->mmXDIBHeader.BMPInfoHeader2.cx;
ulHeight = pMMImgHdr->mmXDIBHeader.BMPInfoHeader2.cy;
if (ulWidth % 4)
    ul4PelWidth = (((ulWidth >> 2) + 1) << 2);
else
     ul4PelWidth = ulWidth;
pVidInfo->mmotHeader.mmXlen = (USHORT) ul4PelWidth;
pVidInfo->mmotHeader.mmYlen = (USHORT) ulHeight;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Write the M-Motion Header.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ulBytesWritten = mmioWrite(pVidInfo->hmmioSS,
                           (PVOID) &pVidInfo->mmotHeader,
                           (ULONG) MMOTION_HEADER_SIZE);
/ * * * * * * * * * * * * * * * * * * * *
  * Check for an error on the write...
  * * * * * * * * * * * * * * * * * * * */
if (ulBytesWritten != MMOTION_HEADER_SIZE)
    return (0);
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  * Flag that MMIOM_SETHEADER has been done. It can only
  * be done ONCE for a file. All future attempts will
  * be flagged as errors.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
pVidInfo->bSetHeader = TRUE;
/*****************************
 * Create copy of MMIMAGEHEADER for future use.
 ********************************/
pVidInfo->mmImgHdr = *pMMImgHdr;
/*******************************
 * Check bitcount, set palette if less than 24.
 ********************************/
if (usBitCount < 24)
   {
   /*****************************
    * Find out how many colors are in the palette.
    ******************************/
   usNumColors = (USHORT)(1 << usBitCount);
   /*****************************
    * Take the RGB2 palette and convert it to an RGB palette
    *     Place the converted palette in MMFILESTATUS struct
    ******************************/
   RGB2_To_RGB (pVidInfo->mmImgHdr.bmiColors,
                (PRGB) &(pVidInfo->rgbPalette),
                usNumColors);
   }
/*******************************
 * We must allocate the buffer.The app will load the
 * buffer on subsequent write calls.
 ********************************/
usPlanes = pVidInfo->mmImgHdr.mmXDIBHeader.BMPInfoHeader2.cPlanes;

/*******************************
 * Determine total Image size
 *******************************
 * Find bits-per-line BEFORE padding and 1bpp or 4bpp pel overflow
 ********************************/
ulImgBitsPerLine = ulWidth * usPlanes * usBitCount;
ulImgBytesPerLine = ulImgBitsPerLine >> 3;
/*******************************
 * Account for extra pels not on an even byte boundary
 *     for 1bpp and 4bpp
 ********************************/
if (ulImgBitsPerLine % 8)
```

```
        ulImgBytesPerLine ++;
      pVidInfo->ulImgPelBytesPerLine = ulImgBytesPerLine;
     /*******************************
       * Ensure the row length in bytes accounts for byte padding.
       *    All bitmap data rows are aligned on LONG/4-BYTE boundaries.
       *    The data FROM an application should always appear in this form
       ******************************/
      usPadBytes = (USHORT) (ulImgBytesPerLine % 4);
      if (usPadBytes)
        ulImgBytesPerLine += 4 - usPadBytes;
      pVidInfo->ulImgPaddedBytesPerLine = ulImgBytesPerLine;
      pVidInfo->ulImgTotalBytes = ulImgBytesPerLine * ulHeight;
     /*******************************
       * Get space for full image buffer.
       ******************************/
      if (DosAllocMem ((PPVOID) &(pVidInfo->lpImgBuf),
                       pVidInfo->ulImgTotalBytes,
                       fALLOC))
         return (MMIO_ERROR);
     /*******************************
       * Set up initial pointer value within RGB buffer & image
       ******************************/
      pVidInfo->lImgBytePos = 0;
      return (sizeof (MMIMAGEHEADER));
      } /* end case of MMIOM_SETHEADER */
```

图 4-16 MMIOM_SETHEADER 情况(MMOTPROC.C)

4.5 CODEC 支持

以下各节分别介绍 Ultimotion IOProc (ULIOT)样例(位于\TOOLKIT\SAMPLES\MM\ULTIMOIO子目录下),包括压缩或解压缩一个数据对象的源代码。ULIOT I/O 过程调用 Ultimotion CODEC 过程压缩原始数字图象,使其以最小的形式占用最少的存储空间。要播放运动视频,ULIOT 样例还可以调用 CODEC 过程的解压缩部分,从压缩的数据重建原始图象。

4.5.1 解压缩

以下各小节分别描述如何解压缩图象,以播放运动视频。解压缩步骤如下:

1. 打开文件,确定要求哪个或哪些 CODEC。

2. 调用 CODEC DLL 文件。

3. 调入并初始化 CODEC 过程。

4. 调用 CODEC,解压缩数据。

5. 使用之后,关闭 CODEC 并释放资源。

4.5.1.1 打开一个图象对象

要回放或解压缩一个图象对象,必须首先打开电影文件,找到压缩类型的四字符代码(FOURCC)并初始化声道信息(如图 4-17 示)。打开路径允许每个流有多种压缩类型。

```
LONG IOProcOpen (PMMIOINFO pmmioinfo, PSZ pszFileName) {
   LONG          rc = MMIO_SUCCESS;        /* Return code .            */
   LONG          lFilePosition;            /* Logical file position .  */
   MMIOINFO      Localmmioinfo;            /* For locally used .       */
   PINSTANCE     pinstance;                /* Local work structure .   */

   if (pmmioinfo == NULL) return MMIO_ERROR;

   if (CheckMem((PVOID)pmmioinfo, sizeof(MMIOINFO), PAG_WRITE))
      return MMIO_ERROR;

  /*******************************************************************/
  /* Validate the open flags for this File Format IOProc             */
  /* (INVALID_OPEN_FLAGS should be defined in the ff.h - file format */
  /* specific header file.)                                          */
  /*******************************************************************/

   if (pmmioinfo->ulFlags & INVALID_OPEN_FLAGS) {
      pmmioinfo->ulErrorRet = MMIOERR_INVALID_ACCESS_FLAG;
      return(MMIO_ERROR);
   }

   ENTERCRITX;
   if ((pinstance = (PINSTANCE) HhpAllocMem (hheap, sizeof (INSTANCE)))
         == NULL) {
      EXITCRIT;
      pmmioinfo->ulErrorRet = MMIOERR_OUTOFMEMORY;
      return(MMIO_ERROR);                  /* Allocate work struct .   */
   }
   EXITCRIT;

   pmmioinfo->pExtraInfoStruct = (PVOID) pinstance;
   pmmioinfo->fccIOProc = HEX_FOURCC_FFIO;/* Set for CODEC loading . */
   ioInstanceInit(pinstance);

  /* Validate read flags before doing read initialization */
```

```
if (( pmmioinfo->ulFlags & MMIO_READ) &&
            !( pmmioinfo->ulFlags & INVALID_READ_FLAGS )) {

  /* IOProc identifies Storage System */

   memcpy (&Localmmioinfo, pmmioinfo, sizeof (MMIOINFO));
   Localmmioinfo.pIOProc = NULL;
   Localmmioinfo.fccIOProc = pmmioinfo->fccChildIOProc;
   Localmmioinfo.ulFlags |= MMIO_NOIDENTIFY; /* Eliminate callbacks */
   Localmmioinfo.ulFlags &= MMIO_ALLOCBUF;  /* Force non-buffered open */

   rc = ioIdentifyStorageSystem(&Localmmioinfo, pszFileName);

   if (rc != MMIO_SUCCESS) {          /* if error,                */
      ioCleanUp(pmmioinfo);
      return(rc);
   }

  /***************************/
  /* Allocate memory for pTempBuffer which is used when   */
  /* IOProcReadInterLeaved is called.                     */
  /***************************/

   if (ENTERCRIT(rc)) {
      ioCleanUp(pmmioinfo);
      return(rc);
   }

   if ((pinstance->pTempBuffer = HhpAllocMem(hheap, DEFAULTBUFFERSIZE))
       == NULL) {
      EXITCRIT;
      ioCleanUp(pmmioinfo);
      return(MMIOERR_OUTOFMEMORY);
   }
   EXITCRIT;
   pinstance->ulTempBufferSize = DEFAULTBUFFERSIZE;
  /***************************/
  /* Open Movie file                                      */
  /***************************/

   if ( pmmioinfo->fccChildIOProc != FOURCC_MEM) {
      Localmmioinfo.cchBuffer = 0;
      Localmmioinfo.pchBuffer = NULL;
   }
   pinstance->hmmioFileHandle = mmioOpen(pszFileName,
```

```
        &Localmmioinfo, MMIO_NOIDENTIFY);
          /* Test file open error.*/
      if (pinstance->hmmioFileHandle <= (HMMIO) 0L) {
         rc = Localmmioinfo.ulErrorRet;
      }
      /****************************/
      /* Call file format specific open routine        */
      /****************************/
      else if ( ! (rc = ffOpenRead(pmmioinfo, pinstance))) {
         if ( ! (rc = ioAddTracksToMovieHeader(pinstance))) {
            /****************************/
            /* Set lLogicalFilePos to a position pass the header     */
            /* block to allow read occurring at the first byte of    */
            /* non-header data.                                      */
            /****************************/
            lFilePosition = ffSeekToDataBegin(pmmioinfo, pinstance);
            if (lFilePosition < MMIO_SUCCESS)
               rc = MMIO_ERROR;
            else
               pinstance->lFileCurrentPosition = lFilePosition;
            }
         }
         if (rc) {
            ioCleanUp(pmmioinfo);
            return(rc);
         }
      }
      /* Validate Write flags before doing initialization */
#ifndef WORKSHOP
    if ((pmmioinfo->ulFlags & (MMIO_READWRITE | MMIO_WRITE)) &&
                 !(pmmioinfo->ulFlags & INVALID_WRITE_FLAGS)) {
      /* Open the movie file */
       memset (&Localmmioinfo, '\0', sizeof(MMIOINFO));
       Localmmioinfo.pIOProc     = NULL;
       Localmmioinfo.fccIOProc   = pmmioinfo->fccChildIOProc;
       if (pmmioinfo->fccChildIOProc != FOURCC_MEM) {
          Localmmioinfo.cchBuffer = 0;
          Localmmioinfo.pchBuffer = NULL;
```

```
      }
      Localmmioinfo.ulFlags |= MMIO_NOIDENTIFY; /* Eliminate callbacks */
      Localmmioinfo.ulFlags &= MMIO_ALLOCBUF;  /* Force non-buffered open. */
                                               /* MMIO may do buffering. */
      pinstance->hmmioFileHandle = mmioOpen(pszFileName, &Localmmioinfo,
                                     MMIO_READWRITE | MMIO_NOIDENTIFY);
      if (pinstance->hmmioFileHandle <= (HMMIO)0L) /* Test file open error. */
         rc = Localmmioinfo.ulErrorRet;
      else
         /* Call file format specific open routine */
         rc = ffOpenWrite(pmmioinfo, pinstance);
      if (rc != 0) {
         ioCleanUp( pmmioinfo);
         return(rc);
      }
   }
#else /* WORKSHOP next */
   if ((pmmioinfo->ulFlags & (MMIO_READWRITE | MMIO_WRITE)) &&
            !(pmmioinfo->ulFlags & INVALID_WRITE_FLAGS)) {
      /* Open the movie file */
      memset (&Localmmioinfo, '\0', sizeof(MMIOINFO));
      Localmmioinfo.pIOProc      = NULL;
      Localmmioinfo.fccIOProc    pmmioinfo->fccChildIOProc;
      Localmmioinfo.ulFlags      = pmmioinfo->ulFlags;
      Localmmioinfo.ulFlags |= MMIO_NOIDENTIFY; /* Eliminate callbacks */
      Localmmioinfo.ulFlags &= MMIO_ALLOCBUF;  /* Force non-buffered open. */
                                               /* MMIO may do buffering. */
      if (!(pmmioinfo->ulFlags & MMIO_CREATE)) {
         rc = ioIdentifyStorageSystem(&Localmmioinfo, pszFileName);
         if (rc != MMIO_SUCCESS) {         /* if error */
            pmmioinfo->ulErrorRet = rc;    /* see IdentifyStorageSystem */
            ioCleanUp(pmmioinfo);
            return(MMIO_ERROR);
         }
         /* Allocate memory for pTempBuffer which is used when */
         /* IOProcReadInterLeaved is called. */
```

```
    if (ENTERCRIT(rc)) {
        ioCleanUp(pmmioinfo);
        return MMIO_ERROR;
    }
    pinstance->pTempBuffer = HhpAllocMem(hheap, DEFAULTBUFFERSIZE);
    if (pinstance->pTempBuffer == NULL) {
        EXITCRIT;
        pmmioinfo->ulErrorRet = MMIOERR_OUTOFMEMORY;
        ioCleanUp(pmmioinfo);
        return MMIO_ERROR;
    }
    EXITCRIT;
    pinstance->ulTempBufferSize = DEFAULTBUFFERSIZE;
}
pinstance->lFileCurrentPosition = 0;
pinstance->hmmioFileHandle = mmioOpen(pszFileName, &Localmmioinfo,
     Localmmioinfo.ulFlags);
if (pinstance->hmmioFileHandle <= (HMMIO)0L)/* Test file open error.*/
   rc = Localmmioinfo.ulErrorRet;
else {
        /* Call file format specific open routine */
    rc = ffOpenWrite(pmmioinfo, pinstance);
    if (rc == 0) {
        if (!(pmmioinfo->ulFlags & MMIO_CREATE)) {
            rc = ioAddTracksToMovieHeader(pinstance);
            if rc == 0) {
                /* Set lLogicalFilePos to a position pass the header */
                /* block to allow read occurring at the first byte */
                /* of non-header data.*/
                lFilePosition = ffSeekToDataBegin(pmmioinfo, pinstance);
                if (lFilePosition < MMIO_SUCCESS) rc = MMIO_ERROR;
                else pinstance->lFileCurrentPosition = lFilePosition;
            }
        }
    }
}
if (rc != 0) {
   pmmioinfo->ulErrorRet = rc;
   ioCleanUp(pmmioinfo);
```

```
            return MMIO_ERROR;
            }
        }
        /* Set up the pathname in the instance structure */
        if (strlen(pszFileName) < CCHMAXPATH) {
            strcpy((PSZ)&(pinstance->szFileName), pszFileName);
            if ((pinstance->szFileName)[1] == ':')
               pinstance->ulEditFlags |= FULLY_QUALIFIED_PATH;
        }
    #endif
        return MMIO_SUCCESS;
    }
```

图 4-17 IOOPEN.C

4.5.1.2 确定 CODEC 过程

一旦得到 FOURCC,便可确定并调用正确的 CODEC 过程了。图 4-18 描述了 ioDetermineCodec 过程,它确定调用哪个 CODEC。该过程首先查询硬件,确定硬件要求什么样的 CODEC 模式;然后,它查询 MMPMMMIO.INI 文件,确定是否有一个 CODEC 过程与从打开的过程中接收到的 FOURCC、压缩类型和容量标志相匹配。如果没有找到默认的 CODEC,则该过程查询 MMPMMMIO.INI 文件,找到一种适用于该硬件的压缩类型。如果 INI 文件所能识别的 CODEC 过程没有一个与 FOURCC 匹配,则代码将搜寻内部 CODEC 表,以调用合适的 CODEC。ULIOT 样例在 ULGDAT.C 文件中提供了一个内部 CODEC 表(如图 4-19 示)。该表在 IOProc 中硬编码,以确定要调用的 CODEC 过程。

```
LONG ioDetermineCodec ( PINSTANCE pinstance,
                        ULONG ulSearchFlags,
                        PCODECINIFILEINFO pcifi)
    {
        LONG            rc = MMIO_SUCCESS; /* Return code of IOProcs call     */
        USHORT          i;                 /* Loop index                      */
        ULONG           ulFlags;
        HPS             hps;               /* Use to query color support      */
        HAB             hab;               /* anchor block                    */
        HMQ             hmq;               /* anchor block                    */
        if (pcifi->ulCapsFlags & CODEC_DECOMPRESS) {
          /* Query the display mode */
           if (ulNumColors == 0) {         /* Get this info once per process  */
              hab = WinInitialize(0);
```

```
//          hmq = WinCreateMsgQueue(hab, 0L);
            hps = WinGetPS(HWND_DESKTOP);
            DevQueryCaps ( GpiQueryDevice(hps),
                           CAPS_COLORS,
                           1L,
                           (PLONG)&ulNumColors);
            WinReleasePS (hps);
//          WinDestroyMsgQueue (hmq);
            WinTerminate (hab);
            }
          /* Set the color depth for the CODEC we want */
          if (ulNumColors == 16)
            pcifi->ulCapsFlags |= CODEC_4_BIT_COLOR;
          else if (ulNumColors > 256)
            pcifi->ulCapsFlags |= CODEC_16_BIT_COLOR;
          else /* 256 and anything else */
            pcifi->ulCapsFlags |= CODEC_8_BIT_COLOR;
          }
      /*************************************************************/
      /* Search for the DEFAULT codec of this type from the MMIO INI file  */
      /*************************************************************/
      pcifi->ulCapsFlags |= CODEC_DEFAULT;  /* Pick default */
      ulFlags = ulSearchFlags |
                MMIO_MATCHFOURCC |
                MMIO_MATCHCOMPRESSTYPE |
                MMIO_MATCHCAPSFLAGS |
                MMIO_MATCHFIRST |
                MMIO_FINDPROC;
      if (!(rc = mmioIniFileCODEC(pcifi, ulFlags))) {
         return (MMIO_SUCCESS);
         }
      /*************************************************************/
      /* If no default, find first one and use it from the MMIO INI file   */
      /*************************************************************/
      pcifi->ulCapsFlags &= CODEC_DEFAULT;
      ulFlags = ulSearchFlags |
                MMIO_MATCHFOURCC |
                MMIO_MATCHCOMPRESSTYPE |
```

```
                MMIO_MATCHCAPSFLAGS |
                MMIO_MATCHFIRST |
                MMIO_FINDPROC;
    /* Match the fourcc, compress type, caps flags */
    if (!(rc = mmioIniFileCODEC(pcifi, ulFlags))) {
       return (MMIO_SUCCESS);
       }
    /*********************************/
    /* Search any internal CODEC tables for the necessary CODEC to load. */
    /* Note: This is used for debugging new CODEC'S that have not been   */
    /* added to the MMPMMMIO.INI file.                                   */
    /*********************************/
    for (i = 0; i < NUMBER_CODEC_TABLE; i++) {

        if ((acifiTable[i].ulCompressType == pcifi->ulCompressType) &&
            ((acifiTable[i].ulCapsFlags & pcifi->ulCapsFlags)
                == pcifi->ulCapsFlags)) {

            *pcifi = acifiTable[i];        /* Copy contents */
            return(MMIO_SUCCESS);
            }
        }

    return(MMIOERR_CODEC_NOT_SUPPORTED);
}
```

图 4-18 ioDetermineCode (IOCODEC.C)

图 4-19 则描述了当 MMPMMMIO.INI **文件中没有显示** CODEC **表时,如何在一个** IOProc **中对** CODEC INI **文件信息结构硬编码。**

```
CODECINIFILEINFO acifiTable[] = {
    {
       sizeof(CODECINIFILEINFO),
       FOURCC_FFIO,
       "ULBDC4",
      /* szDCIODLLName[]-Decompression IOProc DLL name */
       "CodecEntry",
      /* szDCIOProcName[]-Decomp IOProc entry pt proc name */
       UM_VIDEO_COMPRESSION_TYPE_BH146,
      /* ulDecompressionType-ID of each decompression type */
```

```
    0L,
    MMIO_MEDIATYPE_DIGITALVIDEO,
    CODEC_DECOMPRESS +             /* ulCapsFlags-Capabilities Flag */
      CODEC_SELFHEAL +             /* ulCapsFlags-Capabilities Flag */
      CODEC_ORIGIN_UPPERLEFT +     /* ulCapsFlags-Capabilities Flag */
      CODEC_4_BIT_COLOR,           /* ulCapsFlags-Capabilities Flag */
    0,
    0,
    0,
    0,
    0,
    8,
    8,
    0,
  },
  {
    sizeof(CODECINIFILEINFO),
    FOURCC_FFIO,
    "ULBDC8",
      /* szDCIODLLName[]-Decompression IOProc DLL name */
    "CodecEntry",
      /* szDCIOProcName[]-Decomp IOProc entry pt proc name */
    UM_VIDEO_COMPRESSION_TYPE_BH146,
      /* ulDecompressionType-ID of each decompression type */
    0L,
    MMIO_MEDIATYPE_DIGITALVIDEO,
    CODEC_DECOMPRESS +             /* ulCapsFlags-Capabilities Flag */
      CODEC_SELFHEAL +             /* ulCapsFlags-Capabilities Flag */
      CODEC_ORIGIN_UPPERLEFT +     /* ulCapsFlags-Capabilities Flag */
      CODEC_MULAPERTURE +          /* ulCapsFlags-Capabilities Flag */
      CODEC_DIRECT_DISPLAY +       /* ulCapsFlags-Capabilities Flag */
      CODEC_8_BIT_COLOR,           /* ulCapsFlags-Capabilities Flag */
    0,
    0,
    0,
    0,
    0,
    8,
    8,
    0,
```

```
    },
    {
       sizeof(CODECINIFILEINFO),
       FOURCC_FFIO,
       "ULBDC16",
         /* szDCIODLLName[]-Decompression IOProc DLL name */
       "CodecEntry",
         /* szDCIOProcName[]-Decomp IOProc entry point proc name */
       UM_VIDEO_COMPRESSION_TYPE_BH146,
         /* ulDecompressionType-ID of each decompression type */
       0L,
       MMIO_MEDIATYPE_DIGITALVIDEO,
       CODEC_DECOMPRESS +              /* ulCapsFlags-Capabilities Flag */
          CODEC_SELFHEAL +             /* ulCapsFlags-Capabilities Flag */
          CODEC_ORIGIN_UPPERLEFT +     /* ulCapsFlags-Capabilities Flag */
          CODEC_DIRECT_DISPLAY +       /* ulCapsFlags-Capabilities Flag */
          CODEC_16_BIT_COLOR,          /* ulCapsFlags-Capabilities Flag */
       0,
       0,
       0,
       0,
       0,
       8,
       8,
       0,
    },
};
```

图 4-19　CODECINIFILEINFO (ULGDAT.C)

4.5.1.3　调用 CODEC 过程

一旦确定了 CODEC 过程,则调入 CODEC DLL 并加至当前为电影实例调用的相关 CODEC 列表。图 4-20 所示的 ioLoadCodecDLL 过程首先检查 CODEC 过程是不是还没有调入表中,如果没有,则发送 DosLoadModule 函数调用 DLL,并查找入口,然后,将初始化 ffOpenCodec 过程中的 CODEC 过程(如图 4-21 示)。

注意:关于 DosLoadModule 函数的详细信息可参阅《OS/2 PM 参考手册》(OS/2 PM Reference)一书。

```
PCCB ioLoadCodecDLL ( PINSTANCE pinstance,
                      PCODECINIFILEINFO pcifi,
```

```
                    PULONG phCodec )
{
   LONG         rc = MMIO_SUCCESS;    /* Return code of IOProc's call. */
   SZ           szLoadError[260];           /* Error returns. */
   PCCB         pccbNew;
   PCCB         pccb;
   HMODULE      hmod = 0L;
   PMMIOPROC    pmmioproc;
   ULONG        hCodec = 0L;

   /**********************************************************/
   /* Search if the CCB entry has been created for the passed in        */
   /* pszDLLName and pszProcName, if yes, then sets pccb pointer of     */
   /* ptracki to that node.(different track may share a same CCB)       */
   /**********************************************************/
   for (pccb = pinstance->pccbList; pccb; pccb = pccb->pccbNext) {

     if ( !stricmp(pcifi->szDLLName, pccb->cifi.szDLLName)) {
       hCodec = pccb->hCodec;

       if ( !stricmp(pcifi->szProcName, pccb->cifi.szProcName)) {
         /* Proc entry names match */
         return(pccb);
         }
       }
     }/* end loop */

   /**********************************************************/
   /* The above searching cannot find the DCIO node; if a same          */
   /* DLLName was found, query IOProc address directly, else load       */
   /* module then query I/O address before allocates a new pccb node.   */
   /**********************************************************/
   if (DosLoadModule(szLoadError,
                    (ULONG) sizeof(szLoadError),
                    pcifi->szDLLName,
                    &hmod))  {
     /* Load Error_MMIOERR_INVALID_DLLNAME */
     return(NULL);
     }

   if (DosQueryProcAddr(hmod,
                       0L,
                       pcifi->szProcName,
```

```
                          (PFN * )&pmmioproc))  {
      / * Query Error_MMIOERR_INVALID_PROCEDURENAME */
       return(NULL);
       }
  / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  / * The above loading and querying was successful; then create a new    */
  / * node for the DCIO.If HhpAllocMem fails, call DosFreeModule to      */
  / * free DCIO module before returning error.                           */
  / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   if (ENTERCRIT(rc)) {
      return(NULL);
      }
   pccbNew = (PCCB)HhpAllocMem(hheap, (ULONG)sizeof(CCB));

   EXITCRIT;

   IF( ! pccbNew) {
       DosFreeModule(hmod);
      / * Allocate error_MMIOERR_OUTOFMEMORY */
       return(NULL);
       }
  / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  / * Assigns the Decompress IOProc information, which is in record map   */
  / * table, to the new DCIO node.                                        */
  / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   pccbNew->cifi = * pcifi;
   pccbNew->hmodule = hmod;
   pccbNew->pmmioproc = pmmioproc;
   pccbNew->hCodec = 0L;
   pccbNew->ulLastSrcBuf = 0L;
   pccbNew->ulMisc1 = 0L;
   pccbMew->ulMisc2 = 0L;
  / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  / * Adds new node to the beginning of ccb list.                         */
  / * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   pccbNew->pccbNext = pinstance->pccbList;
   pinstance->pccbList = pccbNew;

    * phCodec = hCodec;
   return(pccbNew);
}
```

```
/************START OF SPECIFICATIONS************/
/*                                                                */
/* SUBROUTINE NAME:  ioLoadCodec                                  */
/*                                                                */
/* DESCRIPTIVE NAME; Load a CODEC IOProc and add it the PCCB list */
/*                                                                */
/* FUNCTION: This function loads a CODEC IOProc and adds it to the */
/*   linked list of CODECs currently loaded for this movie instance. */
/*                                                                */
/* NOTES:                                                         */
/*                                                                */
/* ENTRY POINT:  ioLoadCodec                                      */
/*   LINKAGE:    CALL FAR (00:32)                                 */
/*                                                                */
/* INPUT:                                                         */
/*          PINSTANCE pinstance      -Instant structure.          */
/*          PTRACKI ptracki          -Track specific information  */
/*          PCODECINIFILEINFO pcifi  -CODEC ini file information  */
/*                                                                */
/* EXIT-NORMAL:                                                   */
/*          pccb                                                  */
/*                                                                */
/* EXIT-ERROR:                                                    */
/*          0L                                                    */
/*                                                                */
/* SIDE EFFECTS:                                                  */
/*                                                                */
/************END OF SPECIFICATIONS*************/
PCCB ioLoadCodec ( PINSTANCE pinstance,
                   PTRACKI ptracki,
                   PCODECINIFILEINFO pcifi)
{
   LONG      rc = MMIO_SUCCESS; /* Return code of IOProc's call. */
   PCCB      pccbNew;
   ULONG     hCodec = 0L;

   if (pccbNew = ioLoadCodecDLL(pinstance, pcifi, &hCodec)) {
     /*************************************/
     /* Open the Codec IOProc and save the hCodec in the pccb structure   */
     /*************************************/
```

```
        if (rc = ffOpenCodec(pinstance, pccbNew, hCodec, ptracki)) {
           pinstance->pccbList = pccbNew->pccbNext;  /* unlink from list   */
           ioCloseCodec(pccbNew);
           pccbNew = NULL; /* return error condition */
           }
        }
     return(pccbNew);
 }

/************* START OF SPECIFICATIONS *************/
/*                                                                    */
/* SUBROUTINE NAME:  ioFindCodec                                      */
/*                                                                    */
/* DESCRIPTIVE NAME:                                                  */
/*                                                                    */
/* FUNCTION: This function finds a compression/decompression          */
/* control block for this compression type.                           */
/*                                                                    */
/* NOTES: None                                                        */
/*                                                                    */
/* ENTRY POINT:  ioFindCodec                                          */
/*   LINKAGE:    CALL FAR (00:32)                                     */
/*                                                                    */
/* INPUT:                                                             */
/*             PINSTANCE  pinstance-Movie instance structure          */
/*             ULONG      ulCompressType-Compression type             */
/*                                                                    */
/*                                                                    */
/* EXIT-NORMAL:                                                       */
/*             pccb                                                   */
/*                                                                    */
/* EXIT-ERROR:                                                        */
/*             NULL-0L                                                */
/*                                                                    */
/*                                                                    */
/* SIDE EFFECTS:                                                      */
/*                                                                    */
/************* END OF SPECIFICATIONS ***************/
 PCCB ioFindCodec ( PINSTANCE pinstance,
                    ULONG ulCompressType )
 {
```

```
    PCCB pccb;

    for (pccb = pinstance-> pccbList; pccb; pccb = pccb-> pccbNext) {
        if (pccb-> cifi.ulCompressType == ulCompressType)
            return(pccb);
        }
    return(NULL); /* not found */
}
```

图 4-20 ioLoadCodecDLL (IOCODEC.C)

图 4-21 **中的** ffOpenCodec **过程为一个电影实例打开一个** CODEC **实例并创建文件格式指定的头信息**(**包括一个源视频头和一个目标视频头**);**然后**,**它调用** MMIO **服务**,**打开** CODEC。

```
LONG ffOpenCodec ( PINSTANCE pinstance,
                   PCCB pccb,
                   ULONG hCodec,
                   PTRACKI ptracki)
{
    LONG                  rc = MMIO_SUCCESS; /* Return code of IOProc's call. */
    PMMVIDEOHEADER        pmmVideoHdr;       /* Video header node.          */
    PCODECVIDEOHEADER     pcodecvidhdr;
   /********************************************************************/
   /* Fill in necessary information for DC IOProc.                     */
   /********************************************************************/
       /* Get standard track header */
    pmmVideoHdr = (PMMVIDEOHEADER)ptracki-> pTrackHeader;
    pccb-> codecopen.pControlHdr = NULL;
      // FOR AVI, fill in from CODEC SPECIFIC DATA SECTION OF HEADER!
    pccb-> codecopen.pOtherInfo = NULL;

    ENTERCRITX;
   /**********************************************************/
   /* Create Source Video Header                             */
   /**********************************************************/
    if (pcodecvidhdr = (PCODECVIDEOHEADER) HhpAllocMem(hheap, (ULONG)
        sizeof(CODECVIDEOHEADER))) {
        pccb-> codecopen.pSrcHdr = (PVOID)pcodecvidhdr;

        pcodecvidhdr-> ulStructLen = sizeof(CODECVIDEOHEADER);
        pcodecvidhdr-> cx = pmmVideoHdr-> ulWidth;
        pcodecvidhdr-> cy = pmmVideoHdr-> ulHeight;
```

```
pcodecvidhdr->cPlanes = 1;                      /* Hardcoded */
pcodecvidhdr->cBitCount = 16;                   /* Hardcoded */
pcodecvidhdr->ulColorEncoding = MMIO_COMPRESSED;  /* Hardcoded */
/**********************/
/* Create Destination Video Header          */
/**********************/
if (pcodecvidhdr = (PCODECVIDEOHEADER) HhpAllocMem(hheap, (ULONG)
  sizeof(CODECVIDEOHEADER))) {
  pccb->codecopen.pDstHdr = (PVOID)pcodecvidhdr;

  pcodecvidhdr->ulStructLen = sizeof(CODECVIDEOHEADER);
  pcodecvidhdr->cx = pmmVideoHdr->ulWidth;
  pcodecvidhdr->cy = pmmVideoHdr->ulHeight;
  pcodecvidhdr->cPlanes = 1;                        // Hardcoded */
  /***********************/
  /* Initialize the Flags and color encoding        */
  /***********************/
  pccb->codecopen.ulFlags = pccb->cifi.ulCapsFlags &
        (VALID_CODECOPEN_INPUTFLAGS);

  /* Set the color depth for the CODEC we want */
  if (ulNumColors == 16) {
    pccb->codecopen.ulFlags |= CODEC_4_BIT_COLOR;
    pcodecvidhdr->cBitCount = 16;
    pcodecvidhdr->ulColorEncoding = MMIO_PALETTIZED;
    }
  else if (ulNumColors > 256) {
    pccb->codecopen.ulFlags |= CODEC_16_BIT_COLOR;
    pcodecvidhdr->cBitCount = 256;
    pcodecvidhdr->ulColorEncoding = MMIO_RGB_5_6_5;
    }
  else {/* 256 and anything else */
    pccb->codecopen.ulFlags |= CODEC_8_BIT_COLOR;
    pcodecvidhdr->cBitCount = 8;
    pcodecvidhdr->ulColorEncoding = MMIO_PALETTIZED;
    }
  /*********/
  /* Open the Codec
  /*********/
  rc = pccb->pmmioproc( &hCodec,
```

```
                                    MMIOM_CODEC_OPEN,
                                    (LONG)&pccb->codecopen,
                                    0L);
            if ( ! rc) {
               pccb->hCodec = hCodec;
               }
            }
         }
   EXITCRIT;
   return(rc);
}

LONG ffAssociateCodec ( PINSTANCE pinstance,
                        PMMEXTENDINFO pmmextendinfo )
{
   LONG rc = MMIO_SUCCESS;

   return(rc);
}
```

图 4-21　ffOpenCodec (ULCODEC.C)

4.5.2　压缩

以下各小节分别描述如何将原始数字图象压缩成足够小的形式以占用最少的存储空间。压缩支持步骤如下:

1. 确定用哪个 CODEC 过程。
2. 调入 CODEC DLL 文件。
3. 调入并初始化 CODEC 过程。
4. 调用 CODEC 压缩数据。
5. 使用之后关闭 CODEC 并释放资源。

图 4-22 所示的 IOSET.C 文件使用 MMEXTENDINFO 数据结构为源头、目标头及其它信息设定值。

注意: 只能与一个流或道相联。

```
/ * * * * * * * * * * * * START OF SPECIFICATIONS * * * * * * * * * * * */
/ * SOURCE FILE NAME: IOSET.C                                          */
/ *                                                                    */
/ * DESCRIPTIVE NAME: File Format IOProc routine for MMIOM_SET          */
/ *                                                                    */
```

```
/* COPYRIGHT:    IBM - International Business Machines          */
/*             Copyright (c) IBM Corporation 1991, 1992, 1993     */
/*                         All Rights Reserved                    */
/*                                                                */
/* STATUS: OS/2 Release 2.0                                       */
/*                                                                */
/* FUNCTION: This source module contains the set functions.       */
/* NOTES;                                                         */
/*    DEPENDENCIES: none                                          */
/*    RESTRICTIONS: Runs in 32-bit protect mode (OS/2 2.0)        */
/*                                                                */
/* ENTRY POINTS:                                                  */
/*              IOProcSet                                         */
/*                                                                */
/************ END OF SPECIFICATIONS *****************************/

 #include       <stdio.h>
 #include       <string.h>
 #include       <stdlib.h>
 #include       <memory.h>

 #define        INCL_DOS             /* #define INCL_DOSPROCESS.*/
 #define        INCL_ERRORS
 #define        INCL_WIN
 #define        INCL_GPI

 #include       <os2.h>              /* OS/2 headers.           */
 #include       <pmbitmap.h>

 #define        INCL_OS2MM
 #define        INCL_MMIO_CODEC
 #define        INCL_MMIO_DOSIOPROC

 #include       <os2me.h>            /* Multimedia IO extensions. */
 #include       <hhpheap.h>
 #include       <ioi.h>

/************ START OF SPECIFICATIONS ***************************/
/*                                                                */
/* SUBROUTINE NAME: IOProcSet                                     */
/*                                                                */
/* DESCRIPTIVE NAME: Set various conditions in IOProc             */
/*                                                                */
/* FUNCTION:                                                      */
```

```
/*                                                                      */
/* NOTES: None                                                          */
/*                                                                      */
/* ENTRY POINT: IOProcSet                                               */
/*                                                                      */
/*   LINKAGE:     CALL FAR (00:32)                                      */
/*                                                                      */
/* INPUT:                                                               */
/*   PMMIOINFO  pmmioinfo-ptr to instance structure                     */
/*   LONG       lParam1-first parameter                                 */
/*   LONG       lParam2-Second parameter                                */
/*                                                                      */
/*                                                                      */
/* EXIT-NORMAL:                                                         */
/*              MMIO_SUCCESS                                            */
/*                                                                      */
/* EXIT-ERROR:                                                          */
/*              MMIO_ERROR                                              */
/*                                                                      */
/* SIDE EFFECTS:                                                        */
/*                                                                      */
/************ END OF SPECIFICATIONS ************/
LONG IOProcSet ( PMMIOINFO pmmioinfo,
                LONG lParam1,
                LONG lParam2 )
{
   PINSTANCE        pinstance;
   LONG             rc = MMIO_SUCCESS;
   PMMEXTENDINFO    pmmextendinfo = (PMMEXTENDINFO)lParam1;
   PCCB             pccb;
   ULONG            ulCCBCount;
   PCODECASSOC      pcodecassoc;
   ULONG            ulSize;
   PVOID            PtrNextAvail;

   if (rc = ioGetPtrInstance(pmmioinfo,&pinstance))
      return(rc);

   switch(lParam2) {
      /***************/
      /* SET INFO    */
      /***************/
      case MMIO_SET_EXTENDEDINFO;  /* Set extended information */
         if (pmmextendinfo) {   /* error check */
           /************/
```

```
/* Set active track    */
/***********/
if (pmmextendinfo->ulFlags & MMIO_TRACK) {

  if (pmmextendinfo->ulTrackID == (ULONG)MMIO_RESETTRACKS) {
    pinstance->lCurrentTrack = pmmextendinfo->ulTrackID;
    }
  else {
    if (pinstance->ulFlags & OPENED_READONLY) {
      if (ioFindTracki(pinstance, pmmextendinfo->ulTrackID)) {
        pinstance->lCurrentTrack = pmmextendinfo->ulTrackID;
         }
        else {
          pmmioinfo->ulErrorRet = MMIOERR_INVALID_PARAMETER;
          rc = MMIO_ERROR;
          break;
          }
        }

      else if (pinstance->ulFlags &
            (OPENED_READWRITE | OPENED_WRITECREATE)) {
        pinstance->lCurrentTrack = pmmextendinfo->ulTrackID;
        }

      else {
        pmmioinfo->ulErrorRet = MMIOERR_INVALID_PARAMETER;
        rc = MMIO_ERROR;
        break;
        }
      } /* else */
    } /* MMIO_TRACK */

  /*******************************/
  /* Reset all Non_normal reading modes. All request audio      */
  /* and video frames are returned.                              */
  /*******************************/
  if (pmmextendinfo->ulFlags & MMIO_NORMAL_READ) {
    pinstance->ulMode = MODE_NORMALREAD;
    } /* MMIO_NORMAL_READ */

  /*******************************/
  /* Set IOProc into SCAN mode for the active track. Reading     */
  /* will now be done, but only Key frames are returned for      */
```

```
/* video.                                                        */
/*****************************************************************/
else if (pmmextendinfo->ulFlags & MMIO_SCAN_READ) {
   pinstance->ulMode = MODE_SCANREAD;
   } /* MMIO_SCAN_READ */
/*****************************************************************/
/* Set IOProc into REVERSE mode for the active track.            */
/* Reading will now be done, but only Key frames are             */
/* returned for video                                            */
/*****************************************************************/
else if (pmmextendinfo->ulFlags & MMIO_REVERSE_READ) {
   pinstance->ulMode = MODE_REVERSEREAD;
   } /* MMIO_REVERSE_READ */
/*****************************************************************/
/* Associate CODEC information for recording                     */
/*****************************************************************/
if (pmmextendinfo->ulFlags & MMIO_CODEC_ASSOC) {
  /* Don't alow a CODEC association for read only files */
   if (pinstance->ulFlags & OPENED_READONLY) {
      pmmioinfo->ulErrorRet = MMIOERR_INVALID_PARAMETER;
      rc = MMIO_ERROR;
      break;
      }
  /* Can only associate 1 CODEC for record */
   if (pmmextendinfo->ulNumCODECs == 1) {
      if (rc = ioAssociateCodec(pmmioinfo,
                                pinstance,
                                pmmextendinfo->pCODECAssoc)) {
         pmmioinfo->ulErrorRet = rc;
         rc = MMIO_ERROR;
         }
      }
   else {
      pmmioinfo->ulErrorRet = MMIOERR_INVALID_PARAMETER;
      rc = MMIO_ERROR;
      }
   } /* MMIO_CODEC_ASSOC */
} /* pmmextendedinfo */
```

```
    else {/* error-data structure missing */
        pmmioinfo->ulErrorRet = MMIOERR_INVALID_PARAMETER;
        rc = MMIO_ERROR;
        }
    break;
/******************/
/* QUERY BASE AND CODEC INFO     */
/******************/
case MMIO_QUERY_EXTENDEDINFO_ALL:
    /* Query Also CODEC associated info */

    /* Create the array of codecassoc structures to return to caller */
    pcodecassoc = pmmextendinfo->pCODECAssoc; /* Point to beginning */
    for (pccb = pinstance->pccbList; pccb; pccb = pccb->pccbNext) {
        pcodecassoc->pCodecOpen = NULL;
        pcodecassoc->pCCDECIniFileInfo = NULL;
        pcodecassoc++;
        }
    PtrNextAvail = (PVOID)pcodecassoc;

    /* Fill in pointers to the CODECIniFileInfo structures to follow */
    ulSize = 0L;
    pcodecassoc = pmmextendinfo->pCODECAssoc: /* Point to beginning */
    for (pccb = pinstance->pccbList; pccb; pccb = pccb->pccbNext) {

      /* Create and copy CODECINIFILEINFO structure */
      pcodecassoc->pCODECIniFileInfo = (PCODECINIFILEINFO)PtrNextAvail;
      memcpy(pcodecassoc->pCODECIniFileInfo, &pccb->
         cifi, sizeof(CODECINIFILEINFO));
      PtrNextAvail = (PVOID) (((ULONG)PtrNextAvail)
          + sizeof(CODECINIFILEINFO));

       /* Create and copy CODECOPEN structure */
       pcodecassoc->pCodecOpen = PtrNextAvail;
       memcpy(pcodecassoc->pCodecOpen, &pccb->codecopen,
          sizeof(CODECOPEN));
       PtrNextAvail = (PVOID) (((ULONG)PtrNextAvail)
           + sizeof(CODECOPEN));

        /* Create and copy Pointers to structures      */
        /* in the CODECOPEN structure.                  */
        if (pccb->codecopen.pControlHdr) {
           ulSize = *((PULONG)pccb->codecopen.pControlHdr);
           ((PCODECOPEN)pcodecassoc->pCodecOpen)->pControlHdr =
```

```
                (PVOID)PtrNextAvail;
            memcpy(((PCODECOPEN)pcodecassoc->pCodecOpen)->pControlHdr,
                   pccb->codecopen.pControlHdr,
                   ulSize);
            PtrNextAvail = (PVOID) (((ULONG)PtrNextAvail) + ulSize);
            }
        if (pccb->codecopen.pSrcHdr) {
            ulSize = *((PULONG)pccb->codecopen.pSrcHdr);
            ((PCODECOPEN)pcodecassoc->pCodecOpen)->pSrcHdr
               = PtrNextAvail;
            memcpy(((PCODECOPEN)pcodecassoc->pCodecOpen)->pSrcHdr,
                   pccb->codecopen.pSrcHdr,
                   ulSize);
            PtrNextAvail = (PVOID) (((ULONG)PtrNextAvail) + ulSize);
            }
        if (pccb->codecopen.pDstHdr) {
            ulSize = *((PULONG)pccb->codecopen.pDstHdr);
            ((PCODECOPEN)pcodecassoc->pCodecOpen)->pDstHdr
               = PtrNextAvail;
            memcpy(((PCODECOPEN)pcodecassoc->pCodecOpen)->pDstHdr,
                   pccb->codecopen.pDstHdr,
                   ulSize);
            PtrNextAvail = (PVOID) (((ULONG)PtrNextAvail) + ulSize);
            }
        if (pccb->codecopen.pOtherInfo) {
            ulSize = *((PULONG)pccb->codecopen.pOtherInfo);
            ((PCODECOPEN)pcodecassoc->pCodecOpen)->pOtherInfo
               = PtrNextAvail;
            memcpy(((PCODECOPEN)pcodecassoc->pCodecOpen)->pOtherInfo,
                   pccb->codecopen.pOtherInfo,
                   ulSize);
            PtrNextAvail = (PVOID) (((ULONG)PtrNextAvail) + ulSize);
            }
        pcodecassoc++;
    }

/*******************************/
/* QUERY BASE INFO (NOTE: Fall through         */
/* from previous case!)                         */
/*******************************/
case MMIO_QUERY_EXTENDEDINFO_BASE:/* Query only MMEXTENDINFO info */
```

```
        pmmextendinfo->ulStructLen = sizeof(MMEXTENDINFO);
        pmmextendinfo->ulTrackID = (ULONG)pinstance->lCurrentTrack;
       /* pmmextendinfo->pCODECAssoc = NULL; */

       /* Compute ulBufSize for complete information return */
        ulSize = 0L;
        for (pccb = pinstance->pccbList, ulCCBCount = 0;/* Count CCB'S */
             pccb;
             ulCCBCount++, pccb = pccb->pccbNext) {
           ulSize += sizeof(CODECASSOC) + sizeof(CODECOPEN)
                  + sizeof(CODECINIFILEINFO);/* static stuff */
          /* Extract ulStructLen as first field of structure   */
          /* that ptr points to.                                */
           if (pccb->codecopen.pControlHdr) {
              ulSize += *((PULONG)pccb->codecopen.pControlHdr);
              }
           if (pccb->codecopen.pSrcHdr) {
              ulSize += *((PULONG)pccb->codecopen.pSrcHdr);
              }
           if (pccb->codecopen.pDstHdr) {
              ulSize += *((PULONG)pccb->codecopen.pDstHdr);
              }
           if (pccb->codecopen.pOtherInfo) {
              ulSize += *((PULONG)pccb->codecopen.pOtherInfo);
              }
           }
        pmmextendinfo->ulNumCODECs = ulCCBCount;
        pmmextendinfo->ulBufSize = ulSize;
        break;
     /*****/
     /* ERROR */
      *****/
      default:
         pmmioinfo->ulErrorRet = MMIOERR_INVALID_PARAMETER;
         rc = MMIO_ERROR;
         break;
      }/* end switch */
   return(rc);
}
```

图 4-22　IOSET.C

4.5.2.1 将 CODEC 与文件相联

图 4-23 描述如何将一个 CODEC 过程与文件(道)相联。

```
LONG ioAssociateCodec ( PMMIOINFO pmmioinfo,
                        PINSTANCE pinstance,
                        PCODECASSOC pcodecassoc )

    {
       LONG              rc = MMIO_SUCCESS;/ * Return code of IOProc's call. */
       PCCB              pccb;
       ULONG             hCodec:/ * Possibly returned from ioLoadCodecDLL */

      / * Check for NULL pointers */
       if ( ! pcodecassoc- > pCodecOpen || ! pcodecassoc- > pCODECIniFileInfo) {
          return (MMIOERR_INVALID_PARAMETER);
          }

      / * Force the correct values into the CODECINIFILEINFO structure */
       pcodecassoc- > pCODECIniFileInfo- > ulStructLen = sizeof(CODECINIFILEINFO);
       pcodecassoc- > pCODECIniFileInfo- > fcc = pmmioinfo- > fccIOProc;
       pcodecassoc- > pCODECIniFileInfo- > ulCapsFlags | = CODEC_COMPRESS;

      / * Find the codec to load */
       if (rc = ioDetermineCodec(pinstance, 0, pcodecassoc- > pCODECIniFileInfo))
          {
          return(rc);/ * return error */
          }

       else {/ * load and open the compression codec */
          / * * * * * * * * * * * * * * * * * * * * * * * * */
          / * Check for previously installed CODECs.         */
          / * De-install any loaded. Load new one.           */
          / * Allows only 1 CODEC to be loaded at a time.    */
          / * * * * * * * * * * * * * * * * * * * * * * * * */
          if (pinstance- > pccbList) {
             pccb = pinstance- > pccbList;
             pinstance- > pccbList = pccb- > pccbNext;  / * unlink from list */
             ioCloseCodec(pccb);
             }

          / * Load the codec dll */
          if (pccb = ioLoadCodecDLL(pinstance,
                                    pcodecassoc- > pCODECIniFileInfo,
                                    &hCodec)) {

             / * Save the codec open information in the ccb */
```

```
            ((PCODECOPEN)pcodecassoc->pCodecOpen)->ulFlags |= CODEC_COMPRESS;
           /* Force open of compressor */
            if (!(rc = ioInitCodecopen(pccb, (PCODECOPEN)pcodecassoc->
               pCodecOpen))) {
              /* Open the codec */
               if (!(rc = pccb->pmmioproc(&hCodec,
                                              MMIOM_CODEC_OPEN,
                                              (LONG)&pccb->codecopen,
                                              0L))) {
                  pccb->hCodec = hCodec;        /* save handle to codec */
                  }
               }
           /* handle error conditions */
            if (rc) {
               pinstance->pccbList = pccb->pccbNext;  /* unlink from list */
               ioCloseCodec(pccb);
               }
            }
          else {
             rc = MMIO_ERROR;
             }
          }
      return(rc);
  }
```

图 4-23 ioAssociateCodec (IOCODEC.C)

4.5.2.2 为压缩分配内存

上节中的图 4-23 所示的 ioAssociateCodec 过程调用 ioInitCodecopen 过程分配内存。图 4-24 则描述 ioInitCodecopen 过程如何分配内存并如何初始化存在 CODEC 控制块(CCB)中的 CODECOPEN 结构。

```
  LONG ioInitCodecopen ( PCCB pccb,
                         PCODECOPEN pcodecopen)
  {
     ULONG          ulSize;

     ENTERCRITX;
     pccb->codecopen.ulFlags = pcodecopen->ulFlags;
```

```
    /* Create and copy Pointers to structures in CODECOPEN structure */
    if (pcodecopen->pControlHdr) {
        ulSize = *((PULONG)pcodecopen->pControlHdr);
        if (!(pccb->codecopen.pControlHdr = (PVOID)HhpAllocMem(hheap,ulSize)))
            {
             return(MMIO_ERROR);
             }
        memcpy(pccb->codecopen.pControlHdr, pcodecopen->pControlHdr, ulSize);
        }

    if (pcodecopen->pSrcHdr) {
        ulSize = *((PULONG)pcodecopen->pSrcHdr);
        if (!(pccb->codecopen.pSrcHdr = (PVOID)HhpAllocMem(hheap,ulSize)))
            {
            return(MMIO_ERROR);
            }
        memcpy(pccb->codecopen.pSrcHdr, pcodecopen->pSrcHdr, ulSize);
        }

    if (pcodecopen->pDstHdr) {
        ulSize = *((PULONG)pcodecopen->pDstHdr);
        if (!(pccb->codecopen.pDstHdr = (PVOID)HhpAllocMem(hheap,ulSize)))
            {
            return(MMIO_ERROR);
            }
        memcpy(pccb->codecopen.pDstHdr, pcodecopen->pDstHdr, ulSize);
        }

    if (pcodecopen->pOtherInfo) {
        ulSize = *((PULONG)pcodecopen->pOtherInfo);
        if (!(pccb->codecopen.pOtherInfo = (PVOID)HhpAllocMem(hheap,ulSize)))
            {
            return(MMIO_ERROR);
            }
        memcpy(pccb->codecopen.pOtherInfo, pcodecopen->pOtherInfo, ulSize);
        }

    EXITCRIT;
    return(MMIO_SUCCESS);
}
```

图 4-24　ioInitCodecopen (ULCODEC.C)

4.5.2.3 关闭 CODEC 过程

图 4-25 描述了一个如何关闭 CODEC 实例的例子。ioCloseCodec 过程释放与 CODEC 相关的内存。

```
LONG ioCloseCodec ( PCCB pccb )
{
   LONG          rc = MMIO_SUCCESS; /* Return code of IOProc's call. */

   ENTERCRITX;
   if (pccb->codecopen.pSrcHdr) {
      HhpFreeMem(hheap, (PVOID)pccb->codecopen.pSrcHdr);
      }
   if (pccb->codecopen.pDstHdr) {
      HhpFreeMem(hheap, (PVOID)pccb->codecopen.pDstHdr);
      }
   if (pccb->codecopen.pControlHdr) {
      HhpFreeMem(hheap, (PVOID)pccb->codecopen.pControlHdr);
      }
   if (pccb->codecopen.pOtherInfo) {
      HhpFreeMem(hheap, (PVOID)pccb->codecopen.pOtherInfo);
      }
   if (pccb->hCodec) {
      rc = pccb->pmmioproc(&pccb->hCodec,
                           MMIOM_CODEC_CLOSE,
                           0L;
                           0L);
      if (!rc) {
         pccb->hCodec = 0L;
         }
      }
   if (pccb->hmodule) {
//---- DosFreeModule(pccb->hmodule);
      pccb->hmodule = 0;
      }
   HhpFreeMem(hheap,(PVOID)pccb);
   pccb = NULL;
   EXITCRIT;
   return(rc);
}
```

图 4-25 ioCloseCodec（IOCODEC.C）

第 5 章　安装要求

本章论述了如何利用多媒体安装文件(MINSTALL)安装 OS/2 多媒体子系统。MINSTALL 有:

· 媒体控制驱动程序(MCD)

· 流处理器

· I/O 过程

用户可以编控制文件或生成一个安装 DLL 文件来安装子系统。MINSTALL 可以修改 INI 文件和 CONFIG.SYS 文件中的语句,并提供连续安装过程。

多数软件开发人员将利用 MINSTALL 程序的控制文件来安装子系统。MINSTALL 不需要特殊的硬件支持且不会因硬件信息而改变操作系统。因此,如果某些特殊的硬件或操作系统需要此信息,用户需编写一个安装 DLL 文件(包括控制文件)。请参阅“编辑安装 DLL 文件”一节。

5.1　主控制文件

MINSTALL 程序(MINSTALL.EXE)要求特殊的文件信息用来安装子系统。此信息由主控制文件 CONTROL.SCR 提供。主控制文件用来指示安装程序如何安装,安装到何处,如何显示给用户以及需修改哪些系统文件。CONTROL.SCR 通过关键词的说明向 MINSTALL 提供上述信息(参见表 5-1)。

主控制文件必须取名为 CONTROL.SCR 并且必须存储在安装软件包的第一张盘上(软磁盘或光盘)。在安装时,主控制文件中发现的任何错误将记录到 MINSTALL.LOG 文件中(参阅“安装 LOG 文件”一节)。

CONTROL.SCR 文件由头文件及子系统定义两部分组成。头文件部分用来提供安装的一般信息,随后子系统定义部分提供安装软件包中子系统的信息。

5.1.1　CONTROL.SCR 头文件

CONTROL.SCR 头文件包括以下信息:

· 安装软件包的名字

· 生成文件时所需代码页

· 列表控制文件的文件名

· 安装软件包中子系统的数目

· 安装所需媒体单元(光盘或软磁盘)的数目

· 安装时所需媒体单元数

· 源目录及目的目录名

图 5-1 是一个 CONTROL.SCR 头文件的例子,此例在 \ TOOLKIT \ SAMPES \ MM \ CF 目录下。

```
/ * control.scr */

package    = "IBM Multimedia presentation Manager Toolkit/ 2"
codepage   = 437
filelist   = "filelist.tlk"
groupcount = 12
munitcount = 6
medianame = "IBM Multimedia presentation Manager Toolkit/ 2 Installation
Diskette 1"
medianame = "IBM Multimedia presentation Manager Toolkit/ 2 Installation
Diskette 2"
 .
 .
 .
sourcedir = "\ \ "                    = 0
sourcedir = "\ \ lib \ \ "            = 1
 .
 .
 .
destindir = "\ \ mmos2 \ \ "                              = 0
destindir = "\ \ mmos2 \ \ mmtoolkt \ \ lib \ \ "         = 1
 .
 .
 .
```

图 5-1 CONTROL.SCR 头文件

表 5-1 描述了头文件中的关键词。

表 5-1 CONTROL.SCR 头文件关键词

关键词	说　　明
PACKAGE	用一个带双引号的字符串指明安装软件包的名字。例如: PACKAGE = "IBM Multimedia Presentation Manager Toolkit/ 2"
CODEPAGE	指明文件生成时的代码页。例如: CODEPAGE = 437
FILELIST	指明列表控制文件的文件名。此文件由一个文件表组成,由于说明各个文件的特点以及文件存储在软件包中哪张盘上,同时还要说明文件将被拷贝到何处。例如: FILELIST = "FILELIST.TLK"

关键词	说　　明
GROUPCOUNT	指明软件包中子系统的数目。所有的组都将包括在内,包括 0 组(如果存在)。例如: GROUPCOUNT = 10
MUNITCOUNT	指明安装时所需媒体单元(软磁盘或光盘)的数目。这个数必须大于 0,它是软件包存储所需的光盘或磁盘数目。例如: MUNITCOUNT = 6
MEDIANAME	指明媒体名。这里媒体名是磁盘或光盘卷标上的字符串。对于每个媒体单元,关键词必须用双引号的字符串加以说明。这个信息用于安装时提醒用户在需要时插入新盘。例如: MEDIANAME = ″IBM Multimedia Presentation Manager Toolkit/ 2 Installation Diskette 1″
SOURCEDIR	指明源目录名及其相关数值。关键词可用双引号字符串后加一个等号及一个数值的形式来说明。这个数值用于区别各个特殊的目录。此说明可以是空的,这种情况下默认值为 0。路径必须用路径分隔符隔开。例如: SOURCEDIR = ″\ \ LIB \ \ ″= 1
DESTINDIR	指明目的目录名及其相关数值。关键词可用双引号字符串后加一个等号及一个数值的形式来说明。这个数值用于区别各个特殊的目录。此说明可以是空的,在这种情况下默认值为 0。路径必须用路径分隔符隔开。例如: DESTINDIR = ″\ \ MMOS 2 \ \ ″= 0

当用户想生成或修改一个 CONTROL.SCR 头文件时,请注意以下指导信息:

- 用户需将关键词 MUNITCOUNT 的说明放在关键词 MEDIANAME 说明的前面,其它关键词说明无此顺序要求。
- 每个目的目录(DESTINDIR)都必须有一个特定的数值。
- 组可以存在多张盘上,不必只存在一张盘上。
- 用户可以定义任何目录,MINSTALL 将按照关键词 DESTINDIR 的说明产生新的子目录。
- 用户如将安装文件拷贝到另一种不同容量的盘中,则媒体单元数(MEDIACOUNT)将会改变。
- 用户可以按照下面形式在头文件中加注释:用空行或/ * 和 */ 括起来的文本文件。用户不可以使用嵌套注释。
- 用户可以在等号两边使用空格,这里空格是忽略的。
- 用户如想在字串中使用双引号或斜划线,则必须使用换码字符 \ 。
- 可用 \ n 换行。

5.1.2 CONTROL.SCR 子系统定义

CONTROL.SCR 文件中子系统定义部分在头文件后面,由安装软件包中各个子系统

的定义组成,它包括大量信息来说明文件的每个特性。

子系统定义部分信息有下面几种:

· 组号或特性值
· 特性名
· 各组成部分版本
· 所有用于特性安装的文件的大小
· 用于更改 INI 文件和 CONFIG.SYS 文件的控制文件的文件名
· DLL 文件名及其输入点

图 5-2 是一个 CONTROL.SCR 子系统定义部分的例子。

```
ssgroup   = 0
ssname    = "Toolkit. Base"
ssversion = "0.63.0"
sssize    = 13

/*              - 7 = clock   */
ssgroup   = 7
ssname    = "Clock Utility"
ssversion = "0.63.0"
sssize    = 839
ssinich   = "TLKCLOCK.SCR"

/*              - 8 = midiconv   */
ssgroup   = 8
ssname    = "MIDI File Format Converter"
ssversion = "0.63.0"
sssize    = 170
ssinich   = "TLKCONV.SCR"

/*              - 3 = midiio   */
ssgroup   = 3
ssname    = "MIDI IO Procedure"
ssversion = "0.63.0"
sssize    = 475

/*              - 11 = mcistrng   */
ssgroup   = 11
ssname    = "Media Control Interface String Test"
ssversion = "0.63.0"
sssize    = 194
ssinich   = "TLKSTRN.SCR"
```

```
/ *              - 9  = duet1    */
ssgroup   = 9
ssname    = "Duet Player I"
ssversion = "0.63.0"
sssize    = 4854
ssinich   = "TLKDUT1.SCR"

/ *              - 10 = duet2    */
ssgroup   = 10
ssname    = "Duet Player II"
ssversion = "0.63.0"
sssize    = 816
ssinich   = "TLKDUT2.SCR"

/ *              - 12 = inc, lib, and h    */
ssgroup   = 12
ssname    = "Header Files, Include Files and Libraries"
ssversion = "0.63.0"
sssize    = 384
ssconfigch = "TOOLKIT.CH"

/ *              -13 = prog ref    */
ssgroup   = 13
ssname    = "program Reference"
ssversion = "0.63.0"
sssize    = 400
ssinich   = "TLKBOOKR.SCR"

/ *              - 14 = workbook    */
ssgroup   = 14
ssname    = "Workbook"
ssversion = "0.63.0"
sssize    = 150
ssinich   = "TLKBOOKW.SCR"

/ *              - 2 = avcinst    */
ssgroup   = 2
ssname    = "AVC I/O Procedure Installation Utility"
ssversion = "0.63.0"
sssize    = 102
ssinich   = "TLKIOPU.SCR"
```

```
/ *              - 4 = caseconv   */
 ssgroup   = 4
 ssname    = "Case Converter I/ O Procedure"
 ssversion = "0 .63 .0"
 sssize    = 82
```

图 5-2　CONTROL .SCR 子系统定义部分

CONTROL .SCR 子系统定义部分关键词说明见表 5-2。

表 5-2　CONTROL .SCR 子系统定义部分关键词

关键词	说　　明
SSGROUP	指定组,它标志组的开头并赋给组一个值。每组必须有各自的值(0—49)。另外,不同的软件包中的组可以有相同值。这些组在安装选择主窗口中按照组值的升序排列。例如: SSGROUP = 5
SSNAME	指定组名,组名是个 ASCII 码字串,且须用双引号形式。组名可以包含特殊字符且可以转换。组名在安装选择窗口中显示出来。例如: SSNAME = "CD Audio"
SSVERSION	用双引号字串的形式指定各组版本。这个字串须用"dd .dd .dd"(dd 表示数字)的格式描述。任何不是用这种格式说明的版本将转化为这种格式。任何非数字或非·的字符将被认为是零。在第二个点之后的点也将认为是零。例如: SSSIZE = "1 .1 .0"
SSSIZE	指明组中所有文件的大小。它以千字节为单位(500 = 500KB)。这个值用来确定硬盘是否有足够空间来支持安装。如果无法确认组的大小,可以采用一个较大的值。例如: SSSIZE = 1024
SSINICH	指明由 INI 文件更改信息组成的文件的文件名。如无此语句,将无法修改 INI 文件。例如: SSINICH = "ACPAINI .CH"
SSCONFIGCH	指明由 CONFIG.SYS 文件更改信息组成的文件的文件名。如无此语句,将无法修改 CONFIG.SYS 文件。例如: SSCONFIGCH = "ACPACON .CH"
SSCOREQS	用来指明本组运行时所需支持的其它组目录。它确定当前组运行时所依靠的其它组。这些组必须已经安装(如无此语句,则没有相关运行组)。这些相关运行组通过组号来识别。组 A 可以将组 B 作为相关运行组,而组 B 可以没有相关运行组。用户如选择安装了一个组,但没有选择安装它所有的相关运行组,则在安装开始时会出现提示信息。此语句可以在需要时重新说明。例如: SSCOREQS = 1

续表

关键词	说　明
SSICON	指明图形文件文件名。这些文件用于将在安装选择主窗口中显示出来的组。在安装选择窗口中显示的图形文件必须存在第一张安装盘上。如无此语句,将采用一个默认值。例如: SSICON = "ACPA.ICO"
SSDLL	命名安装时将运行的 DLL 文件。此文件将在所有文件拷贝到目的盘后运行,但必须在任何写操作执行之前。如有此语句,则 SSDLLENTRY 语句也必须有。例如: SSDLL = "MY.DLL"
SSDLLENTRY	用双引号字串的形式指明进入 SSDLL 的进入点。如有此语句,则 SSDLL 语句也必须有。例如: SSDLLENTRY = "MyEntry"
SSTERMDLL	命名安装时将运行的 DLL 文件。此文件将在所有文件已拷贝到目的盘且所有写操作已执行之后运行。此语句的作用是让操作显示在多媒体系统中。如有此语句,则 SSTERMDLLENTRY 语句也须有。例如: SSTERMDLL = "MyTERM.DLL"
SSTERMDLLENTRY	指明进入 SSTERMDLL 的进入点名。此名需用双引号字串的形式说明。如有此语句,则 SSTERMDLL 语句也必须有。例如: SSTERMDLL = "MyTermEntry"
SSDLLINPUTPARMS	用双引号字串形式指明安装 DLL 文件所需信息。此信息必须作为 DLL 的一个参数在 SSDLL 或 SSTERMDLL 语句中说明。例如: SSDLLINPUTPRAMS = "5, FILE.NAM, 1.1.0"
SSSELECT	确定安装时子系统的选择。有下面 5 个值。 "ALWAYS"　此组将一直被安装。它只适用于 0 组。具有此值的组在安装选择窗口中不显示出来。 "REQUIRED"　此组被选择安装。如此组以前已安装过,且当前的安装软件包比已安装的软件版本高,用户必须选择安装。如此组以前没有安装过,则用户可以不安装。 "VERSION"　只有当此组已安装过,且当前的安装软件包比已安装软件版本高的情况下,此组才被选择安排,因此用户可不安装此组。 "Yes"　此组不论以前是否安装过都将被选择安装。用户可以决定安装与否,默认时表明不安装。 "No"　此组不被安装,除非用户选择安装它。 "BASENEWER"　此组的文件只在用户机器无软件包或已安装软件比将要安装的软件版本低的情况下被拷贝。 "ONLYNEWER"　用于确定用户不能安装一个低版本软件。属于本组的文件只在已安装软件版本低的情况下被拷贝。如用户机器没有安装软件或已安装软件版本高,则文件不被拷贝。

用户如欲生成或修改 CONTROL.SCR 子系统定义部分文件,请注意以下几点:

· SSGROUP 语句必须是信息块中的第一条。

· 组可以存在不同媒体。

· 信息块中每个语句都必须有确定值。

· 可以在信息块中用空行或/ * … */ 的形式加注释,但不能用嵌套注释。

· 在等号两边可以加空格,这里空格不被考虑。

· 用户如想在字符串中使用双引号或斜划线,则必须用换码字符 \ 。

· 可用 \ n 换行。

5.2 列表控制文件

主控制文件 CONTROL.SCR 通过 FILELIST 语句指明列表控制文件的文件名。此文件将软件包中所有安装文件列表显示,它包括以下内容:

· 文件名

· 文件所存储的媒体单元数

· 文件将被拷贝到的目的目录

· 文件所确定的组

图 5-3 是一个列表控制文件的例子(省略号代表其它的输入)。文件中第一个非注释的记录是在列表控制文件中显示的文件数。例如:145。

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ * This file contains install information . Strings enclosed in C type      */
/ * comments like this line are comment lines . Blank lines are ignored .    */
/ * Non-blank lines will be parsed and extraneous characters will            */
/ * cause errors . First non-comment line must be the total number of        */
/ * files to be installed .                                                  */
/ *                                                                          */
/ * Total number of entries is 145                                           */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   145
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ * All files on the install disks are listed below . Other information      */
/ * is also given, as follows:                                               */
/ *                                                                          */
/ * Disk #     -The number of the disk on which the file resides .           */
/ *             (Ignored if installing from CD-ROM), These are sorted        */
/ *             from 0 to the number of disks, ascending .                   */
/ *                                                                          */
/ * Group #   -The logical group to which the file belongs . Group           */
```

```
/*              starts at 0.                                           */
/*                                                                     */
/* Dest#      -The destination subdirectory into which the file will be */
/*              copied.Dest# starts at 0.                              */
/*                                                                     */
/* Source#    -The source subdirectory from which the file will be     */
/*              copied.                                                */
/*                                                                     */
/* FileName   - The base filename.                                     */
/* sourcedir = "\\"                        = 0                         */
/* sourcedir = "\\lib\\"                   = 1                         */
/* sourcedir = "\\h\\"                     = 2                         */
/* sourcedir = "\\clock\\"                 = 7                         */
/* sourcedir = "\\duet1\\"                 = 9                         */
/* sourcedir = "\\duet2\\"                 = 10                        */
/* sourcedir = "\\mcistrng\\"              = 11                        */
/* sourcedir = "\\inc\\"                   = 13                        */
/* sourcedir = "\\book\\"                  = 16                        */
/* sourcedir = "\\cdmct\\"                 = 17                        */
/* sourcedir = "\\avcinst\\"               = 18                        */
/* sourcedir = "\\caseconv\\"              = 19                        */
/*                                                                     */
/* destindir = "\\mmos2\\"                                      = 0    */
/* destindir = "\\mmos2\\mmtoolkt\\lib\\"                       = 1    */
/* destindir = "\\mmos2\\mmtoolkt\\h\\"                         = 2    */
/* destindir = "\\mmos2\\install\\"                             = 4    */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\clock\\"            = 7    */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\duet1\\"            = 9    */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\duet2\\"            = 10   */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\mcistrng\\"         = 11   */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\cf\\"               = 12   */
/* destindir = "\\mmos2\\mmtoolkt\\inc\\"                       = 13   */
/* destindir = "\\mmos2\\dll\\"                                 = 14   */
/* destindir = "\\mmos2\\help\\"                                = 15   */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\book\\"             = 16   */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\cdmct\\"            = 17   */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\avcinst\\"          = 18   */
/* destindir = "\\mmos2\\mmtoolkt\\samples\\caseconv\\"         = 19   */
/*                                                                     */
/*       groups                                                        */
/*           -2 = AVC I/O Procedure Installation Utility               */
```

```
/*            -3  = MIDI IO Procedure                                          */
/*            -4  = Case Converter I/O Procedure                               */
/*            -7  = Clock Utility                                              */
/*            -9  = Duet Player I                                              */
/*            -10 = Duet Player II                                             */
/*            -11 = MCI String Test                                            */
/*            -12 = Header Files, Include Files and Libraries                  */
/*            -13 = Program Reference                                          */
/*            -14 = Workbook                                                   */
/*                                                                             */
/* Disk # Group Dest # Source # FileName                                       */
/*****************************************************************************/
/*              mmtoolkt\samples\cf     9   14K                                */
     0     0     12     0     "CONTROL.SCR"
     0     0     12     0     "FILELIST.TLK"
     0     0     12     0     "TLKCLOCK.SCR"
     0     0     12     0     "TLKIOPU.SCR"
     0     0     12     0     "TLKCONV.SCR"
  .
  .
  .
      0     0     4     0     "TLKCLOCK.SCR"
      0     0     4     0     "TLKIOPU.SCR"
      0     0     4     0     "TLKCONV.SCR"
      0     0     4     0     "TLKSTRN.SCR"
  .
  .
  .
      0     10     12     0     "TOOLKIT.CH"
/*              mmtoolkt\inc     10   83K                                      */
     0     12     13     13     "ACB.INC"
     0     12     13     13     "AUDIO.INC"
     0     12     13     13     "DCB.INC"

  .
  .
  .
/*              mmtoolkt\lib     11   50K                                      */
     0     12     1     1     "AUDCTL.LIB"
     0     12     1     1     "DSPMGRDL.LIB"
```

```
        0     12     1     1     "LVDP8000.LIB"
  .
  .
  .
/ *            mmtoolkt\h    22  269K                                    */
      0     12     2     2     "ACB.H"
      0     12     2     2     "AUDCTL.H"
      0     12     2     2     "AUDIO.H"
  .
  .
  .
/ *            mmtoolkt\samples\duet2     12  390K                       */
      0     10     10     10     "duet2.c"
      0     10     10     10     "duet2.h"
      0     10     10     10     "duet2.rc"
  .
  .
  .
/ *            mmtoolkt\samples\avcinst     9  102K                      */
      0     2     18     18     "avcinst.dlg"
      0     2     18     18     "avcinst.def"
      0     2     18     18     "avcinst.ipf"
  .
  .
  .
/ *            mmtoolkt\samples\caseconv     8  72K                      */
      0     4     19     19     "convcvsr.c"
      0     4     19     19     "convproc.rc"
      0     4     19     19     "convconv.c"
  .
  .
  .
/ *            mmtoolkt\samples\midiconv     2  44K                      */
      1     8     15     0     "midiconv.hlp"
      1     8      0     0     "midiconv.exe"
/ *            mmtoolkt\samples\mcistrng     14  194K                    */
      1     11     11     11     "mcistrng.c"
      1     11     11     11     "mcistrng.h"
      1     11     11     11     "mcistrng.rc"
  .
  .
  .
```

图 5-3　列表控制文件

表 5-3 解释了列表控制文件中各项。

表 5-3　列表控制文件各项说明

栏　目	解　　释
Media #	指明文件存储的媒体单元(软磁盘或光盘)号。媒体单元从 0 开始排序,这个值将在除硬盘外的所有媒体中使用。此栏须按升序排列。各媒体单元允许有剩余空间。
Group 或 Subsystem #	指明文件所属组。组号必须是一个从零开始计算的确定的整数(这个值在 CONTROL.SCR 文件中通过 SSGROUP 语句确定),它用来确定哪个文件属于被选择安装的组。
Destination #	指明文件将被拷贝到的目录。此目录在 CONTROL.SCR 文件中通过 DESTINDIR 语句确定。这里必须始终有一个确定值(例如,14 表示目录 \ MMOS2 \ DLL)。
Source #	指明源文件目录。此目录在 CONTROL.SCR 文件中通过 SOURCEDIR 语句确定。这里必须始终有一个确定值(例如,1 表示目录 \ LIB)。
File name	指明源文件名,必须用双引号字串的格式。例如"MINSTALL .EXE"。

5.3　更改控制文件

更改控制文件用于存储 CONFIG.SYS 文件和 INI 文件的修改信息。主控制文件通过 SSCONFIGCH 和 SSINICH 关键词语句来确定更改控制文件。

5.3.1　宏支持

在更改控制文件中可以使用宏命令,主控制文件中也可以使用。当使用宏命令时,在安装时可以不指定路径和驱动器名。宏可以完成以下操作:

· 替换文件的目的路径。
· 替换安装目标驱动器的盘符名。
· 替换 CONTROL.SCR 文件中定义的默认目的驱动器及路径名。
· 替换操作系统中起动盘的盘符名。
· 删除指定文件。

表 5-4 是支持的各种宏命令。

表 5-4　支持宏命令

宏	说　　明
destindir = "$ (DELETE) \\ PATH \\ " = number	宏 $ (DELETE)只能用在主控制文件中的 DESTINDIR 语句。相关的数值在列表控制文件中定义,具有此值的文件将从用户机器中删除
$ (DEST)filename	宏 $ (DEST)用来替换文件的整个目的路径名。例如: $ (DEST) CLOCK .EXE　表示 D: \ MMOS2 \ MMTOOLKT \ SAMPLES \ CLOCK \ CLOCK .EXE

续表

宏	说　　明
$(DRIVE)	宏$(DRIVE)用来替换安装目标驱动器的盘符名。(这里不用在宏命令后加冒号)。例如: $(DRIVE)\MMOS2\TEST1.EXE　表示 D:\MMOS2\TEST1.EXE
$(DIR)#	#是CONTROL.SCR文件中说明的目的目录值。此宏可用来替换在主控制文件DESTINDIR语句所说明的路径和驱动器名。例如: $(DIR)4\MINSTALL.LOG　表示 D:\MMOS2\INSTALL\MINSTALL.LOG
$(BOOT)	替换操作系统的启动盘盘符名。(不用在宏命令后加冒号。)例如: $(BOOT)\OS2\NEW.SYS　表示 C:\OS2\NEW.SYS　这儿C:是已安装OS/2系统的驱动器。

5.3.2　CONFIG.SYS更改控制文件

一些多媒体子系统需要在CONFIG.SYS文件中各自说明。CONFIG.SYS更改控制文件通过使用关键词来生成、增加、合并、替换CONFIG.SYS文件中的语句。主控制文件通过SSCONFIGCH语句来确定各个更改控制文件。

表5-5用于说明在CONFIG.SYS文件中使用的关键词的含义。

表5-5　CONFIG.SYS关键词

关键词	说　　明
DEVICE	在CONFIG.SYS文件中添加新的设备语句,等号右边必须是一个双引号的字串。例如: DEVICE = "$(DEST)DEVICE.SYS/ parameters" 这儿可使用宏,一般可用宏$(DEST)。当用户使用宏$(DEST)时,MINSTALL在控制文件中搜索名为DEVICE.SYS的文件。如果搜索成功,$(DEST)则被此文件的路径所替换,最后的结果就将添加到CONFIG.SYS文件中。例如: DEVICE = "D:\SOMEDIR\DEVICE.SYS/ parameters"
MERGE:	合并数据到CONFIG.SYS文件已存在的语句中。例如: MERGE "LIBPATH"= 1 值1是在CONTROL.SCR文件中定义的目的子目录的值。当MINSTALL查找到此子目录,则将此目录添加到当前的LIBPATH语句中(在这个语句中,分号用来分隔各路径,且在行尾表示此行结束)。如文件中无LIBPATH语句,则将产生此语句。如果语句中等号右边不是一个数字,则必须按照下面格式输入: MERGE "SOMEVAR"= "WHOKNOWS" 这里如果"SOMEVAR"已经存在,则"WHOKNOWS"将添加到此语句行尾。如果不存在,

续表

关键词	说　　明
REPLACE	则语句：SOMEVAR = WHOKNOWS 将添加到 CONFIG.SYS 文件中。 注意：参阅 \ TOOLKIT \ SAMPLES \ MM \ CF 目录下的 TOOLKIT.CH 文件。 用户也可将单词"SET"加到第一个双引号字符串中。除非用户在 CONFIG.SYS 文件中另起一行，否则"SET"将被忽略。在另起一行的情况下，"SET"将被加到这一行的开头部分。例如： MERGE "SET SOMEVAR"= "WHOKNOWS" 替换已存在的 CONFIG.SYS 语句。REPLACE 命令后接一个用双引号括起来的变量名。等号右边可以是个数值或字符串。用户可以使用宏。例如： REPLACE "SET MMBASE"= 0 如果变量在 CONFIG.SYS 文件中，那它的值将变为 0。如果变量不存在，则将添加下面语句： SET MMBASE = C：\ MMOS2； 注意：此语句将替换已存在的语句，不能将此语句与路径语句合用。

5.3.3　INI 更改控制文件

INI 更改控制文件可用来完成以下功能：

· 定义桌面系统文件夹中的程序。

· 定义 MMPM2.INI 文件的更改信息。

· 定义其它 INI 文件的更改信息。

5.3.3.1　定义桌面系统文件夹中程序

用户可以使用 WPObject 块来定义一个文件夹或一个将添到文件夹中的程序。WPObject 块调用 OS/2 WinCreateObject 函数，将图象及标题装入桌面系统。此块将间接改变 OS2.INI 文件。当用户想将所生成对象分类时，参阅《OS/2 显示管理器编程指南》一书。

图 5-4 中用 WPObject 块定义一个文件夹。

```
WPObject =
    (
    WPClassName    = "WPFolder"
    WPTitle        = "title"
    WPSetupString = "ICONFILE = $(DEST)icon; OBJECTID = <folderobjid>"
    WPLocation     = "<parentfolderobjid>"
    WPFlags        = wpflags
    )
```

图 5-4　WPObject 块

title　　　　　　　　指明文件夹的标题。

icon	指明显示对象的图像文件的文件名。
<folderobjid>	指明 OS/2 系统用来查找文件夹中各文件的 ID。工作层通过此语句来确定文件夹存在与否。在其它 WPObject 定义中的 WPLocation 部分也可使用此语句。
<parentfolderobjid>	指明将被安装的文件夹中各对象的 ID 值。例如:"<WP_DESKTOP>"工作层桌面系统文件夹的 ID 值。
WPFlags	此值用来说明如果对象存在将采取何种操作。 0——对象存在,不执行操作。 1——如对象存在则替换。 2——如对象存在则修改对象(用给定值修改指定区域)。

下面的例子中,一个名叫 Multimedia Presentation Manager Toolkit/2 的文件夹被装入到桌面系统中。

```
WPObject =
    (
    WPClassName    = "WPFolder"
    WPTitle        = "Multimedia Presentation\nManager Toolkit/2"
    WPSetupString = "ICONFILE = $(DEST)MMTOOLKT.ICO; OBJECTID = <MMPMTLK>"
    WPLocation     = "<WP_DESKTOP>"
    WPFlags        = 2L
    )
```

用户也可以使用 WPObject 块定义一个将装入到文件夹中的文件,如下图所示:

```
WPObject =
    (
    WPClassName    = "WPProgram"
    WPTitle        = "title"
    WPSetupString = "EXENAME = path file; STARTUPDIR = dir; PROGTYPE = PM;
         ICONFILE = $(DEST)icon; [ASSOCTYPE = type;]
         [ASSOCFILTER = filter;] OBJECTID = <pgmobjid>"
    WPLocation     = "<parentfolderobjid>"
    WPFlags        = wpflags
    )
```

title	指明将被显示在父文件夹的对象下的标题。
path	指明.EXE 文件的路径(或用宏代替)。
file	指明.EXE 文件的文件名。
dir	指明启动目录的路径,可以用宏 $(DIR)# 代替(这里 # 为 CONTROL.SCR 文件中确定的目的目录)。
icon	指明描述对象的图象文件名。

type	指明一种或多种关联类型,如“waveform”。这里各个值用逗号隔开。
filter	指明一种或多种关联筛选程序类型,如“ * .wav”。这里各个值用逗号隔开。
<pgmobjid>	指明 OS/ 2 系统用于查找程序的各个对象的 ID 值。安装程序通过它来判断文件存在与否。在其它 WPObject 定义的 WPLocation 部分不能用此语句。
<parentfolderobjid>	此值用来说明如果对象存在将执行何种操作。 0——对象存在,不执行操作。 1——如对象存在,则替换对象。 2——如对象存在,则修改对象(用给定值修改指定部分)。

JoinEA 块: 图 5-5 所示的 JoinEA 块允许用户将一个 EA 文件添加到父文件或目录中。如果此文件或目录在 WPObject 中已经声明,则此块将执行上述功能。

```
JoinEA =
    (
    JoinFileName   = "full path to file"
    JoinEAFileName = "full path to EA file"
    )
```

图 5-5 JoinEA 块

full path to file	指定文件的路径,可以用宏命令。
full path to EA	指定 EA 文件的路径,可以用宏命令。

JoinLongNameEA 块: 图 5-6 所示的 JoinLongNameEA 块允许用户为准备生成的目录指定目录名,此目录名可以超过 8 个字符的标准文件名长度。这个块用于将一个长名(longname)类型的 EA 文件装到目录下。不论何时此目录作为文件夹对象显示时,长名也将显示出来。此语句应在目录生成前使用。

```
JoinLongNameEA =
    (
    JoinLongName       = "longname"
    JoinLongFileName   = "full path to directory"
    JoinEALongFileName = "full path to EA file"
    )
```

图 5-6 JoinLongNameEA 块

longname	指明将被显示的长名(longname)。

full path to directory　　指明目录的路径,支持宏命令使用。

full path to EA file　　指明 EA 文件的路径,支持宏命令使用。

下面是一个用 JoinLongNameEA 块定义,用于 OS/2 多媒体系统目录的长名(LongName)的例子。长名"Sound Bites"将被添加到 CONTROL.SCR 文件中定义的值为"9"的目录中。这里,"9"表示"Sounds"目录。

```
JoinLongNameEA =
    (
    JoinLongName        = "Sound Bites"
    JoinLongFileName    = "$(DIR)9"
    JoinEALongFileName  = "$(DRIVE):\\MMOS2\\INSTALL\\sounds.eas"
    )
```

此文件是生成文件,并不是由 MINSTALL 拷贝或修改的,因此不能在块中用宏 $(DEST)。注意文件将被安放到\MMOS2\INSTALL 子目录下,所有 EA 文件都将拷贝到此目录下。

5.3.3.2　定义 MMPM2.INI 文件更改信息

有些多媒体子系统在 MMPM2.INI 文件中有各自的信息。媒体设备管理器(MDM)通过此文件来维护多媒体各部分或设备的数据库。

下面的块用于改变 MMPM2.INI 文件。

MciInstallDrv 块:图 5-7 所示块允许用户在系统中安装 MCD。

```
MciInstallDrv =
    (
    DrvInstallName       = "internalname"
    DrvDeviceType        = devicetypecode
    DrvDeviceFlag        = deviceflag
    DrvVersionNumber     = "verno"
    DrvProductInfo       = "name2"
    DrvMCDDriver         = "mcdname"
    DrvVSDDriver         = "vsdname"
    DrvPDDName           = "pddname"
    DrvMCDTable          = "mcdtablename"
    DrvVSDTable          = "vsdtablename"
    DrvShareType         = number1
    DrvResourceName      = "resname"
    DrvResourceUnits     = number2
    DrvClassArray[num] =
           (
              (DrvClassNumber = number3)
```

```
        )
    )
```

图 5-7　MciInstallDrv 块

internalname　指明将被安装的设备名称。此名称不允许有雷同,一般由公司名、设备类型及设备模式组成。例如,IBM 公司的用于声霸卡的声波设备可命名为 ibmwavesb01。

devicetypecode　指明设备类型的代码值。这些代码值由 \ TOOLKIT \ H 子目录下的 MCIOS2.H 文件确定。

deviceflag　通过标志位的设定来确定 LONG INT,此语句用于确定设备是否可控。

verno　用" dd .dd .dd "格式确定版本号。这里 dd 是个数字。

name2　指明产品全称,此名可转换。

mcdname　指明由 MCD 组成的 DLL 的文件名。MCD 由 MDM 装入。(MCD—媒体控制驱动器)

vsdname　指明由 VSD(自动说明驱动器)组成的 DLL 的文件名。

pddname　指明 MCD 所使用的物理设备驱动器名。

mcdtablename　指明由 MCD 命令表组成的 DLL 文件名(OS/ 2 多媒体提供标准命令表)。

vsdtablename　指明由 VSD 命令表组成的 DLL 文件名。

number1　指明共享支持类型的代码。这些代码由 \ TOOLKIT \ H 子目录下的 MMDRVOS2.H 文件确定。

resname　指明用于源驱动器管理的管理器名。

number2　指明源盘的最大数。

num　指明下面部分所定义的源类的数目。

number3　指明类所占有的源盘的最大数。

MciInstallAlias 块:

图 5-8 所示块用于指定安装系统的驱动器的转换名。

```
    MciInstallAlias =
        (
        AliasInstallName = "internalname"
        AliasString      = "aliasstring"
        )
```

图 5-8　MciInstallAlias 块

internalname　指明驱动器的内部名。此名在 MciInstallDrv 块中已定义。

aliasstring　　指明驱动器的转换名。

MciInstallConn 块：媒体驱动器的每个操作都必须定义数据输入输出的路径。此路径作为连接器，已定义连接类型，且每个连接类型有相应的连接类型名。图 5-9 所示的 MciInstallConn 块允许用户在系统中安装连接器。

```
MciInstallConn =
    (
    ConnInstallName = "internalname"
    ConnArray [num1]
        (
            (
            ConnType        = num2
            ConnInstallTo = "connto"
            ConnIndexTo     = num3
            )
        )
    )
```

图 5-9　MciInstallConn 块

iternalname	指明连接器所相连的驱动器内部名，此内部名在 MciInstallDrv 块中已定义。
num1	指明系统中输入点的数目。
num2	指明连接类型。连接类型在 \ TOOLKIT \ H 子目录下的 MCIOS2.H 文件中定义。
connto	指明所将连接的驱动器的内部名。
num3	指明用于连接 connto 中定义的驱动器的连接器。

MciInstallExt 块：如在 MCI-OPEN 数据中定义了设备名，但没有定义设备类型，则设备类型由文件的后缀确定。例如，如果 .WAV 与一个内部驱动器名相关联，则当一个有 .WAV 后缀的文件打开时将使用此驱动器。

图 5-10 中的 MciInstallExt 块允许用户定义系统中的媒体控制界面文件的后缀。

```
MciInstallExt =
    (
    ExtInstallName = "internalname"
    ExtArray [num] =
        (
        (ExtString = "string")
        )
    )
```

图 5-10　MciInstallExt 块

internalname　指明与文件后缀相关联的驱动器名。此名在 MciInstallDrv 块中已定义。

num　指明在 string 部分定义的字串的数目。

string　指明合法的文件后缀。

MciInstallParm 块: 图 5-11 中所示 MciInstallParm 块允许用户定义设备的确定参数。此参数用来向用户的 MCD 提供更多的有关驱动器的信息。例如, MciInstallparm 块用于确定使用何种 MIDI 影射来服务于各个定序程序,并输送到 MIDI MCD。

```
MciInstallParm =
    (
    ParmInstallName = "internalname"
    ParmString      = "device specific parameters"
    )
```

图 5-11　MciInstallParm 块

internalname　指明与参数相关联的驱动器名。此驱动器名在 MciInstallDrv 块中定义。

device specific parameters　确定驱动器所需的参数。此参数用于提供更多驱动器的信息。

5.3.3.3　定义其它 INI 文件更改信息

用户可以生成一个 INI 更改控制文件来修改驱动器所需要的任何 INI 文件。例如,用户可以定义 MIDI 影射,初始 OS/2 的 profile 文件,定义外部页面。用户可用 ProfileData 块来完成以上功能。

图 5-12 是 ProfileData 块的示例。

```
ProfileData =
   (
   ini = "inifilename"
   appname = "appname"
   keyname = "keyname"
   dll = "resourcedllname"
   id = resourceid
   )
```

图 5-12　ProfileData 块

inifilename　指明 OS/2 的 INI 文件名。例如:MIDITYPE.INI。

appname　指明设备名的参数值。此设备名用于 INI 文件。

keyname　　　　　指明将被安装的项目名称(变量名或关键名)。

resourcedllname　指明包含 RCDATA 源文件的 DLL 文件名。RCDATA 源文件的 ID 在 resourceid 中确定。

resourceid　　　　确定 RCDATA 源文件的 ID 值(此值将存在 RC 文件中)。ID 值应是一个长整数,例如,120L。

5.4　编辑安装 DLL 文件

用户可以提供 1 - 2 个安装 DLL。这两个 DLL 可以是具有不同进入点的同一个 DLL,也可以是具有相关进入点的不同的文件。可用下列两种方式调用安装 DLL 文件。

· 在全部文件已拷贝,原本文件处理执行前(用 SSDLL 和 SSDLLENTRY 语句实现)调用此 DLL 文件。

· 在全部文件已拷贝,原本文件处理执行后(用 SSTERMDLL 和 SSTERMDLLENTRY 语句实现)调用此 DLL 文件。

各输入点的参数值如下:

HWND(输入)　主处理过程。允许 DLL 为用户接口生成窗口。

PSZ(输入)　　安装软件包的源路径。

PSZ(输入)　　目标驱动器(盘符加冒号,如 d　)

HWND(输入)　MINSTALL 对象窗口处理,用来接收数据,生成媒体控制接口,执行 CONFIG.SYS 操作。

PSZ(输出)　　一个由 DLL 所需的响应文件数据组成的字符串。此字符串是由 DLL 文件生成的 ASCII 码数据。当用户响应由响应文件字符串提供时,MINSTALL 将使用非扩充安装模式实行安装。编码信息传送给 DLL,所有用户的操作将被忽略。

图 5-13 是一个用于定义 DLL 安装文件的例子。

```
ULONG APIENTRY StartMyInstall (HWND hwndOwnerHandle,
                               PSZ pszSourcePath,
                               PSZ pszTargetDrive,
                               PSZ pszMyParms,
                               HWND hwndMinstallHandle,
                               PSZ pszResponseFile);
```

图 5-13　调用安装 DLL 文件

在各种环境下,MINSTALL 都能为安装 DLL 提供多种服务。用户可以在一台没有安装 OS/ 2 多媒体系统的机器上使用 MINSTALL,也可以在安装了 OS/ 2 多媒体系统的机器上使用。运行 MINSTALL 不必在 OS/ 2 多媒体系统环境下。MINSTALL 具有对 MMPM2.INI 文件和 CONFIG.SYS 文件的控制功能以确定当 MINSTALL 运行时没有

更改上述文件。

当 MINSTALL 控制 MMPM2.INI 文件和 CONFIG.SYS 文件时,安装 DLL 或许需要对这两个文件进行读、写操作。MINSTALL 为安装 DLL 提供与上述两个文件的接口。同样,如果 OS/2 多媒体系统已安装且正在运行,则 MINSTALL 所欲更换的文件将被打开,MINSTALL 将这些文件复制到一个临时目录下以备用。

表 5-6 将安装 DLL 可以采用的数据及数据格式加以说明,DLL 可用 WinSendMsg 或 WinPostMsg 传送数据。

表 5-6 安装 DLL 可采用数据

数据	说明
IM.CODEC1INSTALL	使用 ulCodecCompType 字段安装 CODEC。
mp1 = 0;	/ * Not used */
mp2 = MPFROMP (PINSTCODECINIFILEINFO);	/ * Pointer to the INSTIOPROC structure */
IM.CODEC2INSTALL	使用 fceCodecCompType[5]字段安装 CODEC。
mp1 = 0;	/ * Not used */
mp2 = MPFROMP (PINSTCODECINIFILEINFO);	/ * Pointer to the INSTIOPROC structure */
IM.CONFIGDELETE	删除 CONFIG.SYS 文件中的一行。
mp1 = MPFROMP(PCONFIGDATA);	/ * Pointer to the CONFIGDATA structure */
mp2 = 0;	/ * Not used */
IM.CONFIGENUMERATE	从 CONFIG.SYS 文件中输出一行。
mp1 = MPFROMP(PCONFIGDATA);	/ * Pointer to the CONFIGDATA structure */
mp2 = 0;	/ * Not used */
IM.CONFIGMERGE	将数据合并到已有的 CONFIG.SYS 文件中。
mp1 = MPFROMP(PCONFIGDATA);	/ * Pointer to the CONFIGDATA structure */
mp2 = 0;	/ * Not used */
IM.CONFIGNEW	增添新的一行到 CONFIG.SYS 文件。
mp1 = MPFROMP(PCONFIGDATA);	/ * Pointer to the CONFIGDATA structure */
mp2 = 0;	/ * Not used */

数　　据	说　　明
IM_CONFIGQUERYCHANGED mp1 = 0; / * Not used */ mp2 = 0; / * Not used */	如 CONFIG.SYS 已被修改过则返回一个真值:TRUE。
IM_CONFIGREPLACE mp1 = MPFROMP(PCONFIGDATA); / * Pointer to the CONFIGDATA structure */ mp2 = 0; / * Not used */	替换当前的 CONFIG.SYS 文件。
IM_CONFIGUPDATE mp1 = MPFROMP(PCONFIGDATA); / * Pointer to the CONFIGDATA structure */ mp2 = 0 / * Not used */	修改当前的 CONFIG.SYS 文件。
IM_CREATE_WPS_OBJECT mp1 = 0; / * Not used */ mp2 = MPFROMP(PINSTOBJECTDATA); / * Pointer to the INSTOBJECTDATA structure */	安装文件夹及其内容。
IM_DESTROY_WPS_OBJECT mp1 = 0; / * Not used */ mp2 = MPFROMP(HOBJECT); / * This must be the exact OBJECTID with which the object was created */	删除当前文件夹及其内容。
IM_EA_JOIN mp1 = 0; / * Not used */ mp2 = MPFROMP(PINSTEAJOIN); / * Pointer to the INSTEAJOIN structure */	将 EA 文件添加到其父文件中(此文件是先前分开的)。
IM_EA_LONG_NAME_JOIN mp1 = 0; / * Not used */ mp2 = MPFROMP(PINSTEALONGNAMEJOIN); / * A pointer to the structure that contains the long name as an ASCII string, the file or directory name to which the long name is to be applied, and the new EA name */	生成一个具有长名(longname)的 EA 文件,并将其添加到一个文件或目录中。

数　　据	说　　明
IM_LOG_ERROR	输出数据给 MINSTALL.LOG 文件。此数据可以是错误类型数据或只是信息类型数据。
mp1 = MPFROMP((PSZ)pszStatement); /* The text of the message to insert at the end of the MINSTALL.LOG file */ mp2 = 0; /* Not used */	
IM_MCI_EXTENDEN_SYSINFO	mciSendCommand MCI_SYSINFO 的影射。
mp1 = MPFROML(LONG); /* The MCI_SYSINFO extended function desired */ mp2 = MPFROMP(*MCI_SYSINFO_PARMS); /* the SYSINFO structure */	
IM_MCI_SEND_COMMAND	发送 MCI 命令。
mp1 = 0; /* Not used */ mp2 = MPFROMP(PINSTMCISENDCOMMAND); /* Pointer to the INSTMCISENDCOMMAND structure */	
IM_MIDIMAP_INSTALL	安装 MIDI 影射。
mp1 = 0; /* Not used */ mp2 = MPFROMP(PMIDIMAPINSTALLDATA); /* Pointer to the MIDIMAPINSTALLDATA structure */	
IM_MMIO_INSTALL	安装 IOProc。
mp1 = 0; /* Not used */ mp2 = MPFROMP(PINSTIOPROC); /* Pointer to the INSTIOPROC structure */	
IM_QUERYPATH	查寻被拷贝文件的当前目录。
mp1 = PMFROMP(PSZ); /* The name of the file needed */ mp2 = PMFROMP(PSZ); /* The full path to the file */	
IM_SPI_INSTALL	安装流记录信息。
mp1 = 0; /* Not used */ mp2 = MPFROMP(PSZ); /* The fully qualified path of a SPI resource DLL */	

表 5-6 中数据块在 \TOOLKIT\H 子目录下的 MINSTALL.H 文件中已定义。

编辑安装 DLL 文件时请注意以下几点:

· 任何操作如搜寻用户信息若用时超过 1 秒钟,则将鼠标设置在 SPTR_WAIT。

· 在 INI 文件和 CONFIG.SYS 文件已修改，I/O 操作执行之前，需用宏命令 MM_DISPATCHVARS 和 MM_DISPATCHMESSAGS()使用户接口尽可能响应迅速(这儿的宏命令在 MINSTALL.H 文件中已定义)。这种操作十分必要，因为 DLL 将按照 MINSTALL 中数据顺序依次执行。

5.5 安装媒体控制驱动程序

用于在 OS/2 多媒体系统下安装媒体控制驱动程序(MCD)。

1. 生成一个包含 MCD 所需信息的 INI 更改控制文件。这将在 MMPM2.INI 文件中产生必要的变化。当用户已生成一个新的 MCD，用户必须通过新的 MCD 来安装驱动器到 OS/2 多媒体系统。在图 5-14 的例子中，用户将使用 INI 更改控制文件中的块来安装一个驱动器——SoundBlaster Waveform Audio Driver。这个驱动器由一个名为"MyNewMCD"的媒体控制驱动程序驱动。如欲知 INI 更改控制文件中的块的详细功能，请参阅本章 5.5.3 节"定义 MMPM2.INI 文件更改信息"。

```
MciInstallDrv =
    (
    DrvInstallName      = "MyWaveSB01"
    DrvDeviceType       = 7
    DrvDeviceFlag       = 01L
    DrvVersionNumber    = "1"
    DrvproductInfo      = "Sound Blaster Pro MCV"
    DrvMCDDriver        = "MyNewMCD"
    DrvVSDDriver        = "Audioif"
    DrvPDDName          = "SBAUD1$"
    DrvMCDTable         = "MDM"
    DrvVSDTable         = ""
    DrvShareType        = 3
    DrvResourceName     = "SoundblasterW01"
    DrvResourceUnits    = 1
    DrvClassArray[1]    =
         (
             (DrvClassNumber = 1)
         )
    )
    MciInstallParm =
        (
        ParmInstallName    = "MyWaveSB01"
        ParmString         = "FORMAT = 1, SAMPRATE = 22050, BPS = 8, CHANNELS = 2,
                              DIRECTION = PLAY"
```

```
    )

MciInstallConn =
    (
    ConnInstallName = "MyWaveSB01"
    ConnArray[1] =
        (
          (
          ConnType = 3                  /* Wavestream connector */
          ConnInstallTo = "MyAmpMixSB01 /* Connect to ampmixer   */
          ConnIndexTo = 1               /* First connector in
                                           ampmixer             */
          )
        )
    )

MciInstallAlias =
    (
    AliasInstallName = "MyWaveSB 01"
    AliasString = "Wave Audio"
    )

MciInstallExt =
    (
    ExtInstallName = "MyWaveSB01"
    ExtArray[1] =
        (
          (ExtString = "WAV")
        )
    )
```

图 5-14　安装媒体驱动程序时所用块

2. 在多媒体设置记录本中插入外部设置页。参阅 5.8 节“插入外部设置页”。

3. 在主控制文件的 SSINICH 关键词说明中指定用户的 INI 更改控制文件名。请参阅“CONTROL.SCR 子系统定义”一节，那儿提供了一个示例。例如：

```
SSINICH = "MYMCD.SCR"
```

用户可以用 \ TOOLKIT \ SAMPLES \ MM \ CF 子目录下的程序示范做个实验。如果用户想修改 MCD 模块，可以用自己的驱动程序名替换多媒体系统中的驱动程序。例如，用户可以编辑 MMPM2.INI 文件，将 MCDDRIVER = AUDIOMCD 语句改为 MCD-

DRIVE = AUDIOMCT。同样,用户也可以拷贝自己的 MCD 到系统一个目录下,但此目录必须在用户的 CONFIG.SYS 文件的 LIBPATH 语句中已定义(例, \ MMOS2 \ DLL)。当用户完成上述改动后重启系统,OS/ 2 多媒体系统就将采用用户自己的 MCD 而不是系统提供的 MCD。

注意: 当用户完成 MCD 测试后,必须把 MMPM2.INI 文件恢复原样,否则系统将失去它所支持的 MCD 而可能产生各种料想不到的后果。

5.6 安装流处理器

当 MINSTALL 安装一个流处理器时,会出现下列情况:

1. 媒体驱动器将产生一个 SpiGetHandler 函数。

2. 一个流处理器被加载后,同步/ 流管理器将通过 SPI.INI 文件中的 SPCB(stream protocol control blocks)进行查询。

3. 如果流处理器是一个 DLL, SSM 将通过函数 DOSLoadModule 装入并存储此文件;如果流处理器是一个设备驱动程序, SSM 不用函数 DOSLoadModule,因为在系统启动时, CONFIG.SYS 语句中已装入此系统驱动程序。

在 SSM 查看相关的 SPCB 之前, SPCB 必须已装入 SPI.INI 文件中。SPI.INI 文件是在安装多媒体系统时建立的,这个文件包含所有当前 OS/ 2 多媒体系统支持的流处理器及流类的信息。

5.6.1 生成源文件

为了将 SPCB 信息安装到 SPI.INI 文件,用户须先产生一个源文件,此文件包含流处理器将改变或添加到 SPI.INI 文件中的信息。源文件各部分如图 5-15 所示。

```
  #include <os2.h>
  #include <os2me.h>

  #define SPI_RESOURCE1                    (SPI_RESOURCE + 1)
  #define SPI_RESOURCE2                    (SPI_RESOURCE + 2)

  #define SPCBHAND_RCVSYNC_GENSYNC_GENTIME
                       (SPCBHAND_RCVSYNC + SPCBHAND_GENSYNC + SPCBHAND_GENTIME)

 RCDATA SPI_RESOURCE
 BEGIN
   2                       /* number of stream handlers resources */
 END

 RCDATA SPI_RESOURCE1
   BEGIN
```

```
    "TESTSYS\0",                     /* Class name        */
    "R3TEST\0",                      /* Handler name      */
    SH_DLL_TYPE,                     /* PDD or DLL flag   */
    "R3TEST\0",                      /* PDD or DLL name   */
    1,                               /* Number of SPCBs   */
    SPCBSIZE,                        /* Length of SPCB    */
    DATATYPE_GENERIC,                /* Data type         */
    SVBTYPE_NONE,                    /* Sub type          */
    0L,                         /* Internal key      */
    0L,                         /* Data flag         */
    0L,                         /* # of records      */
    1L,                         /* Block size        */
    4096L,                      /* Buffer size       */
    2L,                         /* Min # of buffers  */
    4L,                         /* Max # of buffers  */
    1L,                         /* # empty buffs to start src */
    2L,                         /* # full buffs to start tgt */
    SPCBBUF_NONCONTIGUOUS,      /* Buffer flag */
    SPCBHAND_RCVSYNC,           /* Handler flag */
    0L,                         /* Sync tolerance value */
    0L,                         /* Save sync pulse generation */
    0L,                         /* Bytes/unit of time       */
        0L                      /* MMTIME each unit represents */

  END
RCDATA SPI_RESOURCE2
  BEGIN
    "TESTSYS\0",                /* Class name        */
    "R0TEST\0",                 /* Handler name      */
    SH_PDD_TYPE,                /* PDD or DLL flag   */
    "R0TEST.SYS\0",             /* PDD or DLL name   */
    2,                          /* Number of SPCBs   */
    SPCBSIZE,                   /* Length of SPCB    */
    DATATYPE_ADPCM_AVC,         /* Data type         */
    0L,                         /* Sub type        */
    0L,                         /* Internal key    */
    SPCBDATA_CUETIME,           /* Data flag       */
    0L,                         /* # of records    */
    1L,                         /* Block size      */
    4096L,                      /* Buffer siZe     */
```

```
    10L,                          /* Min # of buffers */
    10L,                          /* Max # of buffers */
    1L,                           /* # empty buffs to start src */
    1L,                           /* # full buffs to start tgt */
    SPCBBUF_NONCONTIGUOUS,              /* Buffer flag  */
    SPCBHAND_RCVSYNC_GENSYNC_GENTIME, /* Handler flag */
    0L,                           /* Sync tolerance value */
    0L,                           /* Save sync pulse generation */
    0L,                           /* Bytes/unit of time */
    0L,                           /* MMTIME each unit represents */
    SPCBSIZE,                     /* Length of SPCB */
    DATATYPE_WAVEFORM,            /* Data type    */
    0L,                           /* Sub type     */
    0L,                           /* Internal key */
    SPCBDATA_CUETIME,             /* Data flag    */
    0L,                           /* # of records */
    1L,                           /* Block size   */
    8192L,                        /* Buffer size  */
    10L,                          /* Min # of buffers */
    10L,                          /* Max # of buffers */
    1L,                           /* # empty buffs to start src */
    1L,                           /* # full buffs to start tgt */
    SPCBBUF_NONCONTIGUOUS,        /* Buffer flag */
    SPCBHAND_RCVSYNC_GENSYNC_GENTIME, /* Handler flag */
    0L,                           /* Sync tolerance value */
    0L,                           /* Save sync pulse generation */
    0L,                           /* Bytes/unit of time */
    0L                            /* MMTIME each unit represents */
END
```

图 5-15　流处理器源文件

源文件已生成后,用户需编一个短程序用来生成包含源文件的 DLL。**图** 5-16 **是一个程序示范,此程序用来产生一个包含图** 5-15 **所示源文件的** DLL。

```
#include <os2.h>
VOID RCSTUB()
{
}
```

图 5-16　短程序

5.6.2 建立包含源文件的 DLL

下面,用户将建立一个包含源文件的 DLL。此 DLL 用于改变 SPI.INI 文件的 profile 语句。用户可通过输入以下命令来建立 DLL:

```
NMAKE/F MAKERES.MAK
```

图 5-17 是一个简单的 .MAK 文件(MAKERS.MAK),此文件用来建立包含图 5-15 所示源文件的 TESTRES.CLL。

```
.SUFFIXES   .com .sys .exe .obj .mbj .asm .inc .def .lrf .crf .ref \
            .lst .sym .map .c .h .lib .msg .pro .txt .cod .cvk
RCDLL = testres
/ * * * * * * * * * * * * * * * * */
/ * Compiler and Tools location    */
/ * * * * * * * * * * * * * * * * */
MSRC      = ..
TOOLS     = ..\ ..\ TOOLS
SHIP_LIB= ..\ ..\ SHIP_LIB
SHIP_H    = ..\ SHIP_H
SHIP_INC= ..\ SHIP_INC
COMMON    = ..\ COMMON
INC       = ..\ ..\ SRC \ INC
H         = ..\ ..\ SRC \ H
H386      = ..\ ..\ SRC \ H386
LIB       = ..\ ..\ SRC \ LIB
LIB386    = ..\ ..\ SRC \ LIB386
/ * * * * * * * * * * * * * * * * */
/ * Definitions for C Compiler     */
/ * * * * * * * * * * * * * * * * */
CCOMP386 = c1386
CFLAGS386 =/ C/ G3 / AS/ W3 / Od/ DLINT_ARGS
CINC386 = -I . -I $ (SHIP_H) -I $ (COMMON) -I $ (H386) -I $ (H386) \ SYS -I $ (H) -I $ (H) \ SYS
/ * * * * * * * * * * * * * * * */
/ * Definitions for linker      */
/ * * * * * * * * * * * * * * * */
LINK386    = link386
LFLAGS386 = $ (LNK_DEBUG) / batch/ map/ nod/ noi/ packcode/ packdata
LIBS386    = $ (NAMELIB) os2386 libc doscalls
/ * * * * * * * * * * * * * * * * * * * * * */
```

```
/* Definitions for RESOURCE compiler          */
/***********************/
RC      = rc
RCINC   = -i $(H) -i $(SHIP_H) -i $(COMMON)

/***************/
/* Object file lists          */
/***************/

RCOBJS     = $(COMMON)\rcstub.obj

/***********************/
/* Inference Rules                            */
/***********************/

.c.obj:
        $(CCOMP386) $(CFLAGS386) $(CINC386) /Fo$(<R).obj $(C_LST) $(<R).c

/***********************/
/* Target Descriptions                        */
/***********************/

!include          "$(H)\common.mak"

all: rc

/***********************/
/* SSMRES.DLL Target Descriptions            */
/***********************/

rc: $(RCDLL).dll

$(RCDLL).dll:    $(RCOBJS) $(RCDLL).rc makeres.mak $(RCDLL).lrf \
                 $(RCDLL).def
        $(LINK386) $(LFLAGS386) @$(RCDLL).lrf
        $(RC) $(RCINC) $(RCDLL).rc $(RCDLL).dll

#
# Make DEF file
#
$(RCDLL).def: makeres.mak
        @echo Creating file <<$(@B).def
LIBRARY $(RCDLL)
DESCRIPTION 'DLL file containing resources'
STUB 'OS2STUB.EXE'
DATA NONE
<<keep
#
# Make link response file
#
```

```
$(RCDLL).lrf   makeres.mak
        @echo Creating file < < $(@B).lrf
$(RCOBJS)
$(RCDLL).dll
$(RCDLL).map $(LFLAGS386)
os2386 libcdll
$(RCDLL).def;
< <keep
```

图 5-17　用于流处理源 DLL 的 .MAK 文件

5.6.3　修改 SPI.INI 文件

由 .MAK 文件建立的 DLL 将被主控制文件, 列表控制文件及 SpiInstall 块用来修改 SPI.INI 文件(参阅 5.1 节“主控制文件”和 5.2 节“列表控制文件”)。下面是需要修改 SPI.INI 文件的文件示例:

· INI 更改控制文件

· CONTROL.SCR 文件输入

· FILELIST.TLK 文件输入

这些文件需同已生成的 TESTRES.DLL 文件放在同一张盘上。

图 5-18 中的例子将告诉用户如何编辑一个名为 TEST.SCR 的 INI 更改控制文件来安装 DLL 文件(TESTRES.DLL)。

```
SpiInstall =
    (
    SpiDllName = "$(DEST)TESTRES.DLL"
    )
```

图 5-18　用于修改 SPI.INI 的 INI 更改控制文件

图 5-19 的例子表明如何在 CONTROL.SCR 文件中确定这个 TEST.SCR 更改控制文件。

```
package    = "SPI.INI Update"
codepage   = 437
filelist   = "TEST.MMI"
groupcount = 2
munitcount = 1
medianame  = "SPI.INI Update Disk 1"
sourcedir  = "\\"                  = 0
destindir  = "\\MMOS2\\DLL\\"      = 2
```

```
destindir  = "\\ MMOS2 \\ INSTALL \\ "    = 4
ssgroup    = 0
ssname     = "Base"
ssversion  = "1 .0 .0"
sssize     = 10
ssgroup    = 1
ssname     = "SPI .INI Update"
ssversion  = "1 .0 .0"
sssize     = 10
ssinich    = "TEST .SCR"
```

图 5-19　用于修改 SPI.INI 的 CONTROL.SCR 输入

图 5-20 中的示例表明如何在 FILELIST.TLK 文件中确定 TEST.SCR 这个 INI 更改控制文件。

```
/ *  Total number of entries is 3                              */

       3

/ *  Disk #  Group #  Dest #  Path              FileName  */
     0         1        2      0                "TESTRES .DLL"
     0         1        4      0                "TEST .SCR"
     0         0        4      0                "TEST .SCR"
```

图 5-20　用于修改 SPI.INI 的 FILELIST.TLK 输入

5 .6 .4　安装流记录

流处理器必须支持能接受 SPCB 信息的 SpIinstallProtocol。例如,假定一个应用程序或媒体驱动程序需安装其它 SPCB,且此 SPCB 具有和已安装的 SPCB 相同的数据类型,那么此应用程序或媒体驱动程序需被赋予一个新的内部关键值,用来区别具有相同数据类型的各个 SPCB。所有默认的 SPCB 将被赋予值 0,其它已安装的具有重叠数据类型的 SPCB 也需要拥有一个不同的内部关键值,因此流处理器必须具有允许上列情况发生的程序。

图 5-21 中的简单程序说明如何安装流记录。

```
RC ShcInstallProtocol(pipparm)
PPARM_INSTPROT pipparm;
{/ *  Start of ShcInstallProtocol */
RC rc = NO_ERROR;                        / *  local return code  */
```

```
int notfound = TRUE;
PESPCB pTempEspcb;
PESPCB pPrevEspcb;

/ * the ESPCB list is under semaphore control */

if ( ! (rc = DosRequestMutexSem(hmtxGlobalData, SEM_INDEFINITE_WAIT)))
  {/ * obtained semaphore */
    if (pipparm -> ulFlag & SPI_DEINSTALL_PROTOCOL)
      {/ * DeInstall */

        / * To Deinstall, Find the spcb,                    */
        / *                Take it off the espcb chain,     */
        / *                Free the espcb memory allocated */

        rc = ERROR_INVALID_SPCBKEY;
        pPrevEspcb = NULL;
        pTempEspcb = pESPCB_ListHead;
        while (ptempEspcb && notfound)
          {/ * Loop thru espcbs */
            if ((pipparm -> spcbkey.ulDataType ==
                         pTempEspcb -> spcb.spcbkey.ulDataType) &&
                (pipparm -> spcbkey.ulDataSubType ==
                         pTempEspcb -> spcb.spcbkey.ulDataSubType) &&
                (pipparm -> spcbkey.ulIntKey ==
                         pTempEspcb -> spcb.spcbkey.ulIntKey))
              {/ * found match */
                notfound = FALSE;
                rc = NO_ERROR;

               / * Take the espcb off the chain */

                if (pPrevEspcb)
                  {
                    pPrevEspcb -> pnxtESPCB = pTempEspcb -> pnxtESPCB;
                  }
                else
                  {
                    pESPCB_ListHead = pTempEspcb -> pnxtESPCB;
                  }
                HhpFreeMem(hHeap, pTempEspcb);
            }/ * found match */
          else
            {/ * try the next espcb in the chain */
```

```
            pPrevEspcb = pTempEspcb;
            pTempEspcb = pTempEspcb -> pnxtESPCB;
          }/ * try the next espcb in the chain */
        }/ * Loop thru espcbs */
      }/ * DeInstall */
    else
      {/ * Install */
      / * If the SPCB already exists then error */
       if (ShFindEspcb(pipparm -> spcbkey))
         {
           rc = ERROR_INVALID_SPCBKEY;
         }
       else
         {/ * OK to add spcb */
          / * Allocate the espcb and put it on the front of the chain */
           pTempEspcb = (PESPCB)HhpAllocMem(hHeap, sizeof(ESPCB));
           if (pTempEspcb)
             {
               pTempEspcb -> spcb = * (pipparm -> pspcb);
               pTempEspcb-> pnxtESPCB = pESPCB_ListHead;
               pESPCB_ListHead = pTempEspcb;
             }
           else
             }
               rc = ERROR_ALLOC_RESOURCES;
             }
         }/ * OK to add spcb */
       }/ * Install */
      DosReleaseMutexSem(hmtxGlobalData);
      }/ * obtained semaphore */
    return(rc);
}/ * End of ShcInstallProtocol */
```

图 5-21　安装操作记录 SHIPROT.C 文件

注意：如欲知 SPCB 中参数的含义，请参阅第 3 章"流处理器"。

5.7 安装 I/O 过程

欲安装一个 I/O 过程,必须先将 I/O 过程接口添加到 MMPMMMIO.INI 文件中,这可以通过编辑 INI 更改控制文件或用 mmioIniFileHalder 函数编程来实现。此 I/O 过程在 I/O 过程表中排列在系统存储 I/O 过程前面。

图 5-22 中所示的 mmioInstall 块允许用户安装 I/O 过程到系统。当 MINSTALL 运行时,MMPMMMIO.INI 文件由 mmioInstall 块中最后的数据确定。

```
mmioInstall =
    (
    mmioFourCC            = "fourcc"
    mmioDllName           = "full path"
    mmioDllEntryPoint     = "entry name"
    mmioFlags             = "flags"
    mmioExtendLen         = "extend length"
    mmioMediaType         = "media type"
    mmioIOProcType        = "file format"
    mmioDefExt            = "default extension"
    )
```

图 5-22　mmioInstall 块

fourcc	指明 I/O 过程的 FOURCC。
full path	指明 IOProc DLL 的文件名及路径。这里,用户可以使用表 5.4 中所列宏命令。
entry name	指明 IOProc DLL 进入点。
flags	设定任何附加的 MMIO 标志位。当此 MMIO 解除时,设为 0。
extend length	指明扩展长度。
media type	指明文件媒体类型。
file format	指明 IOProc 类型,是文件格式还是存储系统 IOProc。
default extension	指明默认的文件后缀。

例如,为安装 AVC 音频 I/O 过程,先生成一个 INI 更改控制文件来确定 mmioInstall 块。

```
mmioInstall =
    (
    mmioFourCC            = "AVCA"
    mmioDllName           = "$(DEST)AVCAPROC.DLL"
    mmioDllEntryPoint     = "AVCAIOProc"
```

```
    mmioFlags          = 0L
    mmioExtendLen      = 16L
    mmioMediaType      = 2L
    mmioIOProcType     = 2L
    mmioDefExt         = ""
    )
```

用户可以通过主控制文件中的 SSINICH 语句来指定用户自己的 INI 更改控制文件名。请参阅本章“CONTROL.SCR 子系统定义”一节中的例子。例如:

```
SSINICH = "BASE1.SCR"
```

位于 \ MMOS2 \ INSTALL 子目录下的 BASE1.SCR 文件中包含有 mmioInstall 块的示例。用户可以在初始化文件(MMPMMMIO.INI)中用 mmioIniFileHandler 函数确定所欲安装的 I/O 过程。

图 5-23 的示例将告诉用户如何使用 mmioIniFileHandler 函数来安装 OS/2 1.3 PM 字节影射 I/O 过程。

```
 # define FOURCC_OS13 mmioFOURCC('O', 'S', '1', '3')

 # pragma linkage(mmioIniFileHandler, system)

void main ()
{
 ULONG rc;
 MMINIFILEINFO mminifileinfo;
 mminifileinfo.fccIOProc = FOURCC_OS13;
 strcpy (mminifileinfo.szDLLName, "OS13PROC");
 strcpy (mminifileinfo.szProcName, "OS13BITMAPIOPROC");
 mminifileinfo.ulExtendLen = 16L;
 mminifileinfo.ulFlags = 0L;
 mminifileinfo.ulMediaType = MMIO_MEDIA_IMAGE;
 mminifileinfo.ulIOProcType = MMIO_IOPROC_FILEFORMAT;
 strcpy (mminifileinfo.szDefExt, "");
 printf ("Installing OS/2 PM Bitmap (V1.3) IOProc \ n");
 rc = mmioIniFileHandler ( &mminifileinfo, MMIO_INSTALLPROC);
 switch (rc)
 {
   case MMIO_SUCCESS
    printf ("Installing Complete \ n");
    break;
   case MMIOERR_INVALID_PARAMETER
```

```
      printf ("Error in this install program \ n");
      break;
    Case MMIOERR_INTERNAL_SYSTEM
      printf ("OS/ 2 MPM System Error \ n");
      break;
    case MMIOERR_NO_CORE
      printf ("Memory unavailable for this IOProc \ n");
      break;
    case MMIOERR_INI_OPEN
      printf ("Unable to access the OS/ 2 MMPMMMIO.INI file \ n");
      break;
    case MMIOERR_INVALID_FILENAME
      printf ("Cannot find the file   OS13PROC.DLL \ n");
      break;
    default
      printf ("Unknown error attempting to install OS/ 2 Bitmap V(1.3) \ n");
      break;

  }
}
```

图 5-23　安装 I/ O 过程

在 MMPMMMIO.INI 文件中安装 I/ O 过程的优点是使应用程序简单明　。用户一启动 OS/ 2 多媒体系统,I/ O 过程就将建立。注意: IOProc 必须储在 DLL 文件中,如果需要,一个 DLL 文件中可以存储多个 IOProc。

5.8　插入外部设置页

下面的操作用来帮助用户如何在多媒体设置记录本中插入外部设置页。这里,设置页的源代码存在多媒体设置程序之外。当用户想插入外部设置页,且这个记录和一个特定的 MCD 无严格的联系时,可以使用这种操作。例如,用于系统或媒体控制接口设备的页码。

注意: 用户如想为一个特殊的 MCD 插入设置页,请参阅第 2 章 2.10 节" 向多媒体设置笔记本中插入页 "。

MDM 通过 MMPM2.INI 文件来维护用于安装多媒体各设备的数据库。MMPM2.INI 文件由 MINSTALL 程序初始化。MMPM2.INI 文件中附加部分将根据安装控制文件中的关键词的信息来初始化,或者按照应用程序要求调用 MDM 来实现。

用户可以编辑一个包含 ProfileData 块的 INI 更改控制文件来定义外部设置页,具体过程如下:

1．编程来生成设置页。程序示例如图 5-24 所示。

```
HWND InsertExamplePage(PMCI_DEVICESETTINGS_PARMS pMCIDevSettings)
{
  HWND  hwndPage;                  /* Page window handle     */
  CHAR  szTabText[CCHMAXPATH];     /* Buffer for tab string  */
  ULONG ulPageId;                  /* Page Identifier        */
/*************************************/
/*    Load a modeless secondary window                    */
/*************************************/
    hwndPage    = WinLoadSecondaryWindow(
                        pMCIDevSettings->hwndNotebook,
                        pMCIDevSettings->hwndNotebook,
                        ExamplePageDlgProc,
                        vhmodMRI,
                        ID_EXAMPLE,
                        (PVOID)pMCIDevSettings);
    if (!hwndPage) return (NULL);
    ulPageId = (ULONG)WinSendMsg( pMCIDevSettings->hwndNotebook,
                       BKM_INSERTPAGE,
                       (ULONG)NULL,
                       MPFROM2SHORT( BKA_AUTOPAGESIZE |
                       BKA_MINOR, BKA_LAST ) );
/*************************************/
/*    Associate a secondary window with a notebook page.  */
/*************************************/
    WinSendMsg (pMCIDevSettings->hwndNotebook, BKM_SETPAGEWINDOWHWND,
          MPFROMP( ulPageId ), MPFROMLONG( hwndPage ) );
/*************************************/
/*    Get Tab Text from DLL                               */
/*************************************/
    WinLoadString(WinQueryAnchorBlock( HWND_DESKTOP ), vhmodMRI,
        (USHORT) IDB_EXAMPLE, CCHMAXPATH, szTabText );
/*************************************/
/*    Set Tab Text                                        */
/*************************************/
    WinSendMsg( pMCIDevSettings->hwndNotebook, BKM_SETTABTEXT,
        MPFROMP( ulPageId ), szTabText );
```

```
    return( hwndPage );
  }
  typedef struct {
     HWND hwndHelpInstance;
  } MMPAGEINFO;
  typedef MMPAGEINFO * PMMPAGEINFO;
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/ *     Modeless secondary window procedure                          */
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  MRESULT EXPENTRY ExamplePageDlgProc (HWND hwnd, USHORT msg,
                                       MPARAM mp1, MPARAM mp2)
  PMMPAGEINFO pMMPageInfo = (PMMPAGEINFO)
                             WinQueryWindowPtr (hwnd, QWL_USER);
    switch (msg) {
       case WM_INITDLG
          / * Initialize The Page */
          pMMPageInfo = (PMMPAGEINFO) malloc (sizeof(MMPAGEINFO));
          WinSetWindowPtr (hwnd, QWL_USER, pMMPageInfo);
          / * Create a Help Instance */
          pMMPageInfo - > hwndHelpInstance = WinCreateHelpInstance( ...);
          break;
       case WM_DESTROY
          / * Clean up page window resources */
            WinDestroyHelpInstance (pMMPageInfo - > hwndHelpInstance);
            free (pMMPageInfo);
            break;
       case WM_COMMAND
          / * Process All Commands */
          return ((MRESULT) FALSE);
          break;
       case MM_TABHELP
          / * Display help for a tab */
          if (pMMPageInfo - > hwndHelpInstance) {
              WinSendMsg(
                  pMMPageInfo- > hwndHelpInstance,
                  HM_DISPLAY_HELP,
                  MPFROMSHORT(WinQueryWindowUShort(hwnd, QWS_ID)),
                  HM_RESOURCEID );
```

```
        }
          break;
         case WM_CLOSE
            return ((MRESULT) FALSE);
            break;
         case WM_HELP:
            if (pMMPageInfo->hwndHelpInstance) {
               WinSendMsg(
                    pMMPageInfo->hwndHelpInstance,
                    HM_DISPLAY_HELP,
                    (MPARAM) mp1,
                    HM_RESOURCEID );
               }
               return ((MRESULT)TRUE);
               break;
         case WM_TRANSLATEACCEL
              return (WinDefWindowProc (hwnd, msg, mp1, mp2));
                break;
         case HM_QUERY_KEYS_HELP:
              return((MRESULT) IDH_HELPFORKEYS);
                break;
         }
         return (WinDefSecondaryWindowProc(hwnd, msg, mp1, mp2));
             }
```

图 5-24　插入页到记录本

2. 如图 5-25 所示,生成一个包含 ProfileData 块的 INI 更改控制文件。

```
ProfileData=
   (
   ini="$(DIR)0\\MMPM.INI"            /* Name of the INI file       */
   appname="STPM_SettingsPage  7"     /* External settings pages */
   keyname="UniqueName"               /* Name of the page           */
   dll="RESDLLNAME"                   /* Resource DLL name          */
   id=33L                             /* Resource ID                */
   )
```

图 5-25　用于 INI 更改控制文件中的 ProfileData 块

这里,外部设置页码用转换名 STPM_SettingsPage: # 存储。 # 相当于 MCI_DEVICETYPE_XXX 的值。 # 值为 0,则说明设置页码与系统相关。外部设置页码的关键名是用来确定各个页码的各自不同的名字。dll 值指明了包含 RCDATA 源文件的 DLL 文件名,其中 RCDATA 文件的 ID 在 id 部分确定。id 字段指明 RCDATA 源文件的 ID 值,此值应为一个长整数,例如,33L。

对于 INI 更改控制文件,用户需在安装时装入一个源 DLL。数据项的值应由 DLL 文件的模块名及输入点组成(此 DLL 中插入外部设置页)。在图 5-26 所示的例子中,此值为“MakePage, InsertExamplePage”。

```
RCDATA  33
BEDIN
  "MakePage, InsertExamplePage"
END
```

图 5-26　在 MM 设置中插入外部设置页的 RC 文件

这种书写格式从存储在 DLL 中的 RCDATA 源文件里装入大量数据,并用 OS/2 中的 PrfWriteProfileData 函数将数据输出到标准的 OS/2 INI 文件中。ProfileData 块在本章“定义其它 INI 文件更改信息”一节中已详细讨论过。

注意:用这种方式记录的输入点可用于各种媒体控制接口设备类型,因此,如果一个设置页码将运用于某种特定的设备时,则用于处理此设备的函数也将随之而变化。

3. 在 CONTROL.SCR 文件中定义第二步所生成的 INI 更改控制文件。

4. 用户若编辑一个安装 DLL,外部设置页码可以通过调用 OS/2 的 PrfWriteProfileData 函数来安装。如图 5-27 所示。

```
hini = PrfOpenProfile("C  \mmos2\mmpm.ini");
PrfWriteProfileData (hini,
                    "STPM_SettingsPage  7"         /* Appname      */
                    "UniqueName",                  /* Keyname      */
                    "MakePage, InsertExamplePage", /* Value        */
                    "strlen("MakePage, InsertExamplePage") + 1);
                                                   /* Value length */
PrfCloseProfile(hini);
```

图 5-27　PrfWriteProfile 函数

如想了解 PrfWriteProfile 函数的更多信息,请参阅《OS/2 显示管理器编程参考》一书。

5.9　安装 LOG 文件

安装时 MINSTALL 将生成一个名为 MINSTALL.LOG 的安装 LOG 文件。此文件

记录了安装时可能出现错误的信息。当用户安装时,LOG 文件在启动盘的根目录下是缺省的,用户按下安装选择窗口中的 INSTALL 键时,此文件就将拷贝到目的目录下。此文件可以用 PRINT 命令打印或用 TYPE 命令查看。如用户欲在屏幕上查阅此文件,可输入如下命令:

TYPE d \ MMOS2 \ INSTALL \ MINSTALL .LOG

这儿 d 是选择安装的目的驱动器。

如果安装时出现问题,MINSTALL 将在屏幕上显示有关信息。用户若光凭这些信息还不能解决问题,则可以通过察看 LOG 文件来确定应如何操作。

用户如觉得需要调用 IBM 支持,请在调用前先打印 LOG 文件。IBM 个人系统需要用 LOG 信息来帮助解决安装时的问题。

注意:当安装成功时用户可看到一条祝贺信息,但当用户察看 LOG 文件时,会发现文件中报告有不少错误。这意味着错误在安装时已经解决,用户可以不用处理这些错误信息。

图 5-28 是一个成功安装的 MINSTALL.LOG 文件。

```
Begin parsing master file-Z \ RT \ CONTROL .SCR .
Z \ RT \ CONTROL .SCR parsed successfully .
Begin parsing file list-Z \ RT \ MASTERH3 .RT
Z \ RT \ MASTERH3 .RT parsed successfully .
The following file was copied successfully D \ MMOS2 \ README
The following file was copied successfully D \ MMOS2 \ MINSTALL .EXE
 .
 .
 .
The following file was copied successfully D \ MMOS2 \ INSTALL \ WAVEFILE .EAS
Loaded DLL D \ MMOS2 \ DLL \ MMSND .DLL and calling entry point InstallMMSound .
Returned from DLL D \ MMOS2 \ DLL \ MMSND .DLL .
Loaded DLL D \ MMOS2 \ DLL \ QRYUM .DLL and calling entry point LocateUM .
An extended Sysinfo failed call with dwItem = 1024 .
An extended Sysinfo failed call with dwItem = 4096 .
Completed updating MMPM2 .INI with Digital Video Player devices .
Returned from DLL D \ MMOS2 \ DLL \ QRYUM .DLL .
Loaded DLL D \ MMOS2 \ DLL \ QRYCD .DLL and calling entry point LocateCD .
An extended Sysinfo failed call with dwItem = 4 .
 .
```

```
.
.
An extended Sysinfo failed call with dwItem = 4.
Returned from DLL D  \MMOS2\DLL\QRYCD.DLL.
Loaded DLL D  \MMOS2\DLL\QRYAD.DLL and calling entry point LocateMAUDIO.
Completed updating CONFIG.SYS with M-Audio devices
An extended Sysinfo failed call with dwItem = 1024.
An extended Sysinfo failed call with dwItem = 1024.
Completed updating MMPM2.INI with M-Audio devices.
Returned from DLL D  \MMOS2\DLL\QRYAD.DLL.
Starting to parse the file 'D  \MMOS2\INSTALL\BASE.SCR'.
Successfully parsed the file 'D  \MMOS2\INSTALL\BASE.SCR'.
Starting to parse the file 'D  \MMOS2\INSTALL\SMVINI.SCR'.
Successfully parsed the file 'D  \MMOS2\INSTALL\SMVINI.SCR'.
Starting to parse the file 'D  \MMOS2\INSTALL\BASECONF.CH'.
Successfully parsed the file 'D  \MMOS2\INSTALL\BASECONF.CH'.
Starting to parse the file 'D  \MMOS2\INSTALL\SMVCONF.CH'.
Successfully parsed the file 'D  \MMOS2\INSTALL\SMVCONF.CH'.
Loaded DLL D  \MMOS2\DLL\ITERM.DLL and calling entry point ITermEntry.
Returned from DLL D  \MMOS2\DLL\ITERM.DLL.
Loaded DLL D  \MMOS2\DLL\WEPMPINT.DLL and calling entry point WepmPlusAdd.
Returned from DLL D  \MMOS2\DLL\WEPMPINT.DLL.
```

图 5-28　MINSTALL.LOG 文件示例

附录 A　流处理器模块定义

以下信息描述了由 OS/ 2 多媒体系统提供的流处理器的高级设计和操作。被描述的处理器包括:

· 音频流处理器 DLL 与短设备驱动器
· MIDI 影射流处理器 DLL
· 文件系统流处理器 DLL
· 内存流处理器 DLL
· CD-DA 流处理器 DLL
· CD-ROM XA 流处理器 DLL

A .1　音频流处理器

音频流处理器经常被用来控制音频数据的流式传输。在某些情况下, 音频流处理器充当了音频数据的消费者, 而在另外的情况下却扮演了生产者的角色。在其中的任何一种情况下音频数据总是数字化的, 某些时候是压缩的而某些时候则是非压缩的。这些情况中的每一种都将被详细讨论于后文中。除了这些讨论以外, 还包括关于这一模块与同步/ 流管理器之间的交互作用的信息。

该模块被作为一个运行在第 3 环的动态链接库来执行, 就像运行在第 0 环(系统核心特许级) 的短设备驱动器模块(PDD) 一样。

运行于第 3 环的音频流处理器可以调用第 3 环的 DLL 音频 CODEC 来协助压缩以及解压缩音频数据。第 3 环的音频流处理器通过 VSD 中的 DDCMD 接口与音频设备驱动程序进行通信。(关于 VSD DDCMD 接口的定义请查阅《OS/ 2 多媒体编程参考》一书。)

现行的音频设备驱动程序使用标准音频设备驱动器接口(DDCMD) 与第 0 层的短设备驱动程序模块进行通信。

A .1 .1　外部接口描述

第 3 环的音频流处理器外部接口的描述如下:

· 文件名　AUDIOSH .DLL
· 处理器名　AUDIOSH$
· 处理器类　AUDIOSH
· DD/ DLL　DLL
· 源　　当从音频设备记录音频数据时该流处理器可以作为源。当使用这一部件来记录数字音频时, 即是人们所说的音频源流处理器。音频信号被传送给系统并且由音频硬件设备进行数字化, 所得的数据随后经由一组流缓冲区被传送到消费者

流处理器。该流处理器的输出可以有几种不同的形式，诸如非压缩的 16 位 PCM 以及几种压缩 ADPCM 格式。

· 目标　当从源播放音频数据时该流处理器可以作为目标。当使用该流处理器回放记录好的音频样本时，它就充当了音频目标流处理器。数字音频数据将被通过同步/ 流管理器从源流处理器(例如文件系统流处理器)进行流式传输。这些数据是通过由同步/ 流管理器分配和管理的若干流缓冲区来传送的。

这些流缓冲区的准确数目和大小取决于被流式传输的数据类型。决定流缓冲区大小和数目所需的信息被包含在对应特定流类型的流协议控制块(Stream Protocol Control Block，简称 SPCB)中。

该流处理器支持几种不同格式(流类型)的音频数据，诸如 16 位 PCM，AVC 的 8 位 ADPCM，CD-ROM XA 的 8 位 ADPCM 等等。

除了控制音频数据的回放以外，该流处理器在音频流回放的基础上产生同步脉冲。因此，该流处理器在同步关系中可以是主设备也可以是从设备。应用程序可以建立两个流之间的同步关系并且定义同步脉冲粒度，然后同步脉冲将被音频流处理器尽可能地从重复间隔产生，该间隔以定义在数据类型的 SPCB 中的同步粒度为基础。

A.12　设备控制块

第 3 环的音频流处理器支持两种设备控制块(Device Control Blocks，简称 DCB)。DCB 被用来将音频设备驱动程序与对应于这一流实例的流处理器联系起来。DCB 被作为对 SpiCreateStream 的调用中的一个参数来传送。

第 3 环的音频流处理器也支持 VSD 设备控制块(VSD-DCB)。VSD-DCB 标识了被第 3 环的音频流处理器所使用的 VSD 动态链接库，它被作为对 SpiCreateStream 的调用中的一个参数来传送。

```
/************************
 *
 * DCB_AUDIOSH - Device Control Block for the
 *               Audio Stream Handler.
 *               (*** PACKED STRUCTURE ***)
 ***********************/
typedef struct _dcb_audiosh{/* dcb_audiosh - Device Control Block */
    ULONG     ulDCBLen;                 /* Length of structure          */
    SZ        szDevName[MAX_SPI_NAME];  /* Device driver name           */
    ULONG     ulSysFileNum;             /* File handle number - From an */
                                        /* audio_init call              */
    } DCB_AUDIOSH;
typedef struct _dcb_audiosh FAR *PDCB_AUDIOSH;
```

图 A-1　设备控制块

```
/*************************************
 *
 * VSD_DCB - VSD Device Control Block
 *
 *         This structure will allow stream handlers to use the VSD DLL
 *         by using by the additional fields in the structure.
 *
 *             (*** PACKED STRUCTURE ***)
 ************************************/
typedef struct _VSD_DCB {/* vsd_dcb - VSD Device Control Block      */
    ULONG    ulDCBLen;                /* Length of structure           */
    SZ       szDevName[MAX_SPI_NAME]; /* Device driver name            */
    ULONG    ulSysFileNum;            /* File handle number            */
    ULONG    hvsd;                    /* Handle to VSD instance        */
    PFN      pfnvsdEntryPoint;        /* Address of VSD entry point    */
    ULONG    ulReserved1;             /* Reserved for system           */
    ULONG    ulReserved2;             /* Reserved for system           */
    } VSD_DCB;
typedef VSD_DCB FAR *PVSD_DCB;
```

图 A-2　VSD 设备控制块

A.1.3　相关控制块

音频流处理器不支持任何相关控制块。

A.1.4　被支持的隐式事件(EVENT_IMPLICIT_TYPE)

下列对应音频流处理器的隐式(EVENT_IMPLICIT_TYPE)事件是受支持的:

1. EVENT_ERROR

错误代码将包含在 ulStatus 字段中。能被这个流处理器产生并返回的可能错误代码有:

- ERROR_INVALID_BUFFER_RETURNED
- ERROR_DEVICE_OVERRUN
- ERROR_STREAM_STOP_PENDING

2. EVENT_PLAYLISTCUEPOINT

这一事件仅当该流处理器是一个目标并且它从同步/流管理器的缓冲区表中查找到一个尾接提示点时才会产生。该事件将在数据被使用 MMIO 写到文件中以后被通报。PLAYL_EVCB 的 ulMessageParm 字段将被演播表指令中提供的消息所填充,而 mmtimeStream 将不会被填充。

A.1.5 被支持的显式事件

以下对应音频流处理器的显式事件是受支持的:

1.EVENT_CUE_TIME

2.EVENT_CUE_TIME_PAUSE

3.EVENT_DATAUNDERRUN

A.1.6 不被支持的显式事件

以下对应音频流处理器的显式事件是不受支持的:

1.EVENT_CUE_DATA

2.EVENT_DATAOVERRUN(不支持)

A.1.7 被支持的流处理器命令

以下的流处理器命令(SHC)是受支持的。关于这些 SHC 命令及其错误返回代码请查阅《OS/2 多媒体编程参考》一书。

1.SHC_ASSOCIATE

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_INVALID_REQUEST

2.SHC_CLOSE

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

3.SHC_CREATE

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_STREAM_CREATION

· ERROR_INVALID_STREAM

· ERROR_DEVICE_NOT_FOUND

· ERROR_INVALID_BLOCK

· ERROR_INVALID_SPCBKEY

4.SHC_DESTROY

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_INVALID_STREAM

5. SHC_DISABLE_EVENT

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_INVALID_EVENT

6. SHC_DISABLE_SYNC

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

7. SHC_ENABLE_EVENT

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_TOO_MANY_EVENTS

· ERROR_INVALID_EVENT

8. SHC_ENABLE_SYNC

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_INVALID_FLAG

· ERROR_INVALID_STREAM

· ERROR_STREAM_NOTMASTER

· ERROR_INVALID_NUMSLAVES

9. SHC_ENUMERATE_PROTOCOLS

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_INSUFF_BUFFER

10. SHC_GET_PROTOCOL

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_INVALID_SPCBKEY

11. SHC_GET_TIME

可能的返回代码有:

· NO_ERROR

· ERROR_INVALID_FUNCTION

· ERROR_INVALID_STREAM

12 . SHC_INSTALL_PROTOCOL

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_INVALID_SPCBKEY
- ERROR_INVALID_RESOURCES

13 . SHC_NEGOTIATE_RESULT

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION

14 . SHC_SEEK

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_STREAM_NOT_SEEKABLE
- ERROR_DATA_ITEM_NOT_SEEKABLE

15 . SHC_START

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_INVALID_STREAM

16 . SHC_STOP

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_INVALID_STREAM

A.1.8 被支持的基本流协议控制块(SPCB)

音频流处理器支持以下的 SPCB:

1 . DATATYPE_WAVEFORM

子类:

- WAVE_FORMAT_1M08
- WAVE_FORMAT_1S08
- WAVE_FORMAT_1M16
- WAVE_FORMAT_1S16
- WAVE_FORMAT_2M08
- WAVE_FORMAT_2S08
- WAVE_FORMAT_2M16

· WAVE_FORMAT_2S16
· WAVE_FORMAT_4M08
· WAVE_FORMAT_4S08
· WAVE_FORMAT_4M16
· WAVE_FORMAT_4S16
· WAVE_FORMAT_8M08
· WAVE_FORMAT_8S08
· WAVE_FORMAT_8M16
· WAVE_FORMAT_8S16

2. DATATYPE_ALAW

子类:

· ALAW_8B8KM
· ALAW_8B11KM
· ALAW_8B22KM
· ALAW_8B44KM
· ALAW_8B8KS
· ALAW_8B11KS
· ALAW_8B22KS
· ALAW_8B44KS

3. DATATYPE_MULAW

子类:

· MULAW_8B8KM
· MULAW_8B11KM
· MULAW_8B22KM
· MULAW_8B44KM
· MULAW_8B8KS
· MULAW_8B11KS
· MULAW_8B22KS
· MULAW_8B44KS

4. DATATYPE_ADPCM_AVC

子类:

· ADPCM_AVC_VOICE
· ADPCM_AVC_MUSIC
· ADPCM_AVC_STEREO
· ADPCM_AVC_HQ

5. DATATYPE_MIDI

子类:

· SUBTYPE_NONE

6 . DATATYPE_SPV2

子类:

· SPV2_BPCM
· SPV2_PCM
· SPV2_NONE

A.1.9 流处理器限制

音频流处理器受系统内存的限制。

A.2 MIDI 影射流处理器

MIDI 影射流处理器被用来影射 MIDI 数据。该流处理器并不与音频设备驱动程序直接接口而基本上是对 MIDI 数据进行过滤的"过滤器"流处理器。

该模块被作为运行在第 3 环上的一个 OS/2 动态链接库来执行。

A.2.1 冲洗过滤器流群组

过滤器流群组要求一些附加的步骤来正确地擦除其内容。在一个作为样例的流群组中,一个主流通过过滤器处理器与从流相连接,则首先将需要向主流发送一个停止冲洗消息。当主流停止事件被接收到时,第二个停止冲洗消息必须紧接着被发送到从流中。总之,停止冲洗消息必须被分别发送到每一个流中。关于 SHC_STOP 命令消息请查阅《OS/2 多媒体编程参考》一书。

A.2.2 应用程序和媒体驱动程序效能

为了获得最佳特性,每一个应用程序和媒体驱动程序应当具有以下效能:

· SEEK　查找源流并且接着查找目标流。(如果你使用了另外的方法则流时间将是不正确的。)
· STOP DISCARD　停止目标流并且等待所有的停止事件。接下来停止源流并等待该事件。如果一个流处于 EOS 处,则已被停止的错误流将被返回。在源流上等待事件的失败将导致死锁。
· STOP FLUSH　停止源流并等待停止事件。接下来,它停止目标流并等待停止事件。在源流上等待事件的失败将造成死锁。
· STOP PAUSE　总是在同一时刻暂停所有的流。暂停一个流而保持另一个流运行将造成死锁。
· START　启动源流并接着启动目标流,小的 MIDI 文件将在源流上产生一个快速 EOS。不要企图在没有首先启动源流的情况下 START 或 START PREROLL 一个目标流。这样做的企图将导致死锁。
· START PREROLL　启动(非预滚动)源流并接下来 START PREROLL(开始预滚动)目标流。如果你企图 START PREROLL 所有的流,则将造成死锁。

· DESTROY　在同一时刻破坏所有的流。不要试图破坏一个流并接着继续使用另一个流。

· CREATE 与 ASSOCIATE　所有的流必须在任何流能够接受命令之前被创建并联结。不要企图在后来再加入另一个流。

A.2.3　外部接口描述

对 MIDI 影射流处理器外部接口的描述如下:

· 文件名　MISH.DLL

· 处理器名　MISH

· 处理器类　MIDISYS

· PDD/ DLL　DLL

· 源和目标　该流处理器是一个过滤器。因此,在同一时刻它既是源流又是目标流。它从源流消费 MIDI 数据并且产生影射化的 MIDI 数据来加入目标流或流群中。MIDI 影射流处理器可以有一个源流(主流)。

　该流处理器不产生或接收同步脉冲,但它能被包括在一个同步群中。事实上,MIDI 影射的执行是通过将输入流(主流)和输出流(从流)组合起来形成一个同步群。一个流将被创建来对应每一输出端口。MIDI 同步群可以被作为一个群来启动、停止以及查找,只需在每个这样的调用中使用“Slaves”标志。

A.2.4　设备控制块

MIDI 影射流处理器不支持设备控制块。

A.2.5　相关控制块

MIDI 影射流处理器支持以下显示于图 A-3 中的相关控制块。

```
/ * * * * * * * * * * * * * * * * * * * * * * * * *
 * MISH - MIDI stream handler port-stream table ACB
 * * * * * * * * * * * * * * * * * * * * * * * * * */
 # define ACBTYPE_MISH      0x0005L  / * MIDI port-stream table          */
typedef struct _acb_MISH    / * acbmish - MIDI Assoc.Control Block   */
    {
    ULONG     ulACBLen;              / * Length of structure                    */
    ULONG     ulObjType;             / * ACB_MISH                               */
    HSTREAM   hstreamDefault;        / * Default hstream to use when            */
                                     / * mapper is turned off.                  */
    ULONG     ulReserved1;
    ULONG     ulReserved2;
    ULONG     ulNumInStreams;
```

```
    HSTREAM   hstreamInMAX[_PORTS];/ * Array of Input streams              */
    ULONG     ulNumOutStreams;
    HSTREAM   hstreamOut[MAX_PORTS];/ * Array of Output streams            */
                                  / * The index into the array is          */
                                  / * the source channel for that          */
                                  / * stream .                             */
    } ACB_MISH;
/ * * * * * * * * * * * * * * * * * * * * * * * * *
 * MISH - MIDI stream handler SET ACB
 * * * * * * * * * * * * * * * * * * * * * * * * */
 # define ACBTYPE_SET        0x0006L  / * MIDI set function               */
 typedef struct _acb_set    / * acbset - Set Assoc .Control Block     */
    {
    ULONG     ulACBLen;             / * Length of structure              */
    ULONG     ulObjType;            / * ACB_SET                          */
    ULONG     ulFlag;               / * Set flag                         */
    ULONG     ulTempo;              / * Not used .                       */
    } ACB_SET;
/ * ulFlag defines    */
 # define MIDI_MAP_ON        0x0000L  / * turn mapping function on       */
 # define MIDI_MAP_OFF       0x0001L  / * turn mapping function off      */
```

图 A-3　MIDI 影射流处理器相关控制块(ACB)

A.2.6　被支持的隐式事件(EVENT_IMPLICIT_TYPE)

以下对应 MIDI 影射流处理器的隐式事件(EVENT_IMPLICIT_TYPE)是受支持的:

1 . EVENT_ERROR

ulStatus 字段中将包含错误代码。可以被该流处理器产生并返回的可能错误代码有:

· ERROR_INVALID_BUFFER_RETURNED

· ERROR_DEVICE_OVERRUN

· ERROR_STREAM_STOP_PENDING

A.2.7　被支持的显式事件

没有显式事件为 MIDI 影射流处理器所支持。

A.2.8　被支持的流处理器命令

以下流处理器命令(SHC)是受支持的。关于这些 SHC 命令及其错误返回代码请查阅《OS/ 2 多媒体编程参考》一书。

1 . SHC_ ASSOCIATE

可能的返回代码有:

· NO_ ERROR

· ERROR_ INVALID_ FUNCTION

· ERROR_ INVALID_ STREAM(**传送的** hstream **或** hid **或两者均无效**)

· ERROR_ INVALID_ OBJTYPE(**只有** ACBTYPE_ MMIO **是受支持的**)

· ERROR_ INVALID_ BUFFER_ SIZE(ulAcbLen **小于所需**)

· ERROR_ STREAM_ NOT_ STOP(**流必须被停下来进行联结**)

2 . SHC_ CLOSE

可能的返回代码有:

· NO_ ERROR

· ERROR_ INVALID_ FUNCTION

3 . SHC_ CREATE

可能的返回代码有:

· NO_ ERROR

· ERROR_ INVALID_ FUNCTION

· ERROR_ STREAM_ CREATION

· ERROR_ INVALID_ STREAM

· ERROR_ DEVICE_ NOT_ FOUND

· ERROR_ INVALID_ BLOCK

· ERROR_ INVALID_ SPCBKEY

· ERROR_ ALLOC_ RESOURCES

4 . SHC_ DESTROY

可能的返回代码有:

· NO_ ERROR

· ERROR_ INVALID_ FUNCTION

· ERROR_ INVALID_ STREAM

5 . SHC_ ENUMERATE_ PROTOCOLS

可能的返回代码有:

· NO_ ERROR

· ERROR_ INVALID_ FUNCTION

· ERROR_ INSUFF_ BUFFER

6 . SHC_ GET_ PROTOCOL

可能的返回代码有:

· NO_ ERROR

· ERROR_ INVALID_ FUNCTION

· ERROR_ INVALID_ SPCBKEY

7 . SHC_ INSTALL_ PROTOCOL

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_INVALID_SPCBKEY
- ERROR_ALLOC_RESOURCES

8.SHC_NEGOTIATE_RESULT

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_INVALID_STREAM(**传送的** hstream **或** hid **或两者均无效**)

9.SHC_SEEK

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_STREAM_NOT_SEEKABLE
- ERROR_DATA_ITEM_NOT_SEEKABLE
- ERROR_INVALID_STREAM(**传送的** hstream **或** hstream **与** hid **两者无效**)
- ERROR_NOT_SEEKABLE_BY_TIME
- ERROR_STREAM_NOT_STOP

10.SHC_START

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_INVALID_STREAM(**传送的** hstream **或** hstream **与** hid **两者无效**)
- ERROR_DATA_ITEM_NOT_SPECIFIED(**流必须被联结**)

11.SHC_STOP

可能的返回代码有:

- NO_ERROR
- ERROR_INVALID_FUNCTION
- ERROR_INVALID_STREAM(**传送的** hstream **或** hstream **与** hid **两者无效**)
- ERROR_STREAM_NOT_STARTED

A.2.9 被支持的基本流协议控制块

MIDI **影射流处理器支持以下的流协议控制块**(SPCB)。

1.DATATYPE_MIDI

A.2.10 流处理器限制

受系统内存的限制。

A.3　文件系统流处理器

文件系统流处理器以实时应用程序的名义向或从文件系统设备(局部或远程)传输数据。该处理器的独特性在于它利用 MMIO 子系统与各种各样的设备进行接口,诸如硬盘驱动器、软盘驱动器、CD-ROM 驱动器、WORM 驱动器等等。这些设备可以被物理地安装在本地系统硬件中,或者可以被通过服务器计算机上的局域网来访问。

在脚本(例如来源于 RIFF 文件的波形音频)回放中,文件系统流处理器使用 MMIO 来执行指定数据文件的输入输出,并且接着执行流式处理来维持连续可用的供给,数据流随后被注入目标流,例如波形音频流处理器。该处理器并非完全运行于实时模式,但是它必须支持连续的数据流式传输。它也不支持同步控制,因为文件系统设备并不是实时设备。

A.3.1　外部接口描述

对于文件系统流处理器外部接口的描述如下:

· 文件名　　FSSH.DLL

· 处理器名　FSSH

· 处理器类　FILESYS

· PDD/ DLL　DLL

· 源　　该流处理器可以是流之源。

· 目标　该流处理器可以是流之目标。

A.3.2　设备控制块

无。

A.3.3　相关控制块

该处理器支持 ACBTYPE_MMIO 类型的相关控制块。

```
/ * * * * * * * * * * * * * * * * * * * * * * * *
  * FSSH - File System Stream Handler MMIO Object ACB
  * * * * * * * * * * * * * * * * * * * * * * * * */
 # define ACBTYPE_MMIO     0x0001L      / * MMIO object              */

typedef struct _acb_mmio               / * acbmmio - MMIO ACB       */
    {
    ULONG   ulACBLen;                  / * Length of structure      */
    ULONG   ulObjType;                 / * ACB_MMIO                 */
    HMMIO   hmmio;                     / * Handle of MMIO object    */
    } ACB_MMIO;
```

A.3.4 被支持的隐式事件(EVENT_IMPLICIT_TYPE)

· 以下对应文件系统流处理器的隐式(EVENT_IMPLICIT_TYPE)事件是受支持的:

1. EVENT_ERROR

错误类型将被设置在事件控制块的 ulFlag 字段中。将被报告的三种错误类型为:

(1) 临时错误——TEMPORARY_ERROR 0x0000L

在流式传输期间发生的一个错误,但是流处理器或 SSM 或流处理器与 SSM 两者都能继续进行流式传输。

(2) 可恢复错误——RECOVERABLE_ERROR 0x0001L

该错误发生时要求流被停止。如果合适的话应用程序可以重新启动流。

(3) 不可恢复错误——NONRECOVERABLE_ERROR 0x0002L

发生严重错误造成流处理器停止该流。流不能被重启动。应用程序必须发送一个对 SpiDestroyStream 的调用。

错误代码将包含在 ulStatus 字段中。没有错误代码被该流处理器产生并返回。

另外,从下列 API 来的错误可以被返回:

· DosGetInfoBlock
· SMHEntryPoint SMH_NOTIFY
· mmioRead(如果是源,扩展错误代码被包含于 mmioGetLastError 中)
· mmioWrite(如果是目标,扩展错误代码包含于 mmioGetLastError 中)

2. EVENT_PLAYLISTCUEPOINT

该错误仅当流处理器是一个目标且它从同步/流管理器的缓冲区表中找到一个尾接提示点指示符时才会发生。该事件将在数据被使用 MMIO 写到文件系统以后被通报。PLAYL_EVCB 的 ulMessageParm 字段将被提供于演播表指令中的消息所填充。mmtimeStream 字段将不被填充。

A.3.5 被支持的显式事件

文件系统流处理器不支持任何显式事件。

A.3.6 被支持的流处理器命令

以下流处理器命令(SHC)是受支持的。关于这些 SHC 命令及其错误返回代码的描述请查阅《OS/2 多媒体编程参考》一书。

1. SHC_ASSOCIATE

注意:在流可以被启动之前,流要求文件被打开并与流相联结。在没有停止流的情况下重新联结一个新的对象是不允许的。

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 或 hid 或两者均无效)
· ERROR_INVALID_OBJTYPE(仅支持 ACBTYPE_MMIO)

· ERROR_INVALID_BUFFER_SIZE(ulAcbLen 小于所需)
· ERROR_STREAM_NOT_STOP(流必须被停下来进行联结)

从以下 API 返回的代码也可被返回:

· DosRequestMutexSem

2 . SHC_CREATE

注意: 如果被发送的 Spcbkey 不与已安装的任何协议相匹配, 则将使用最后被安装的带 DATATYPE_GENERIC 的流协议控制块。

可能的返回代码有:

· ERROR_INVALID_SPCBKEY
· ERROR_ALLOC_RESOURCES

从以下 API 返回的代码也可被返回:

· DosRequestMutexSem
· DosCreateThread

3 . SHC_DESTROY

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 两者均无效)

从以下 API 返回的代码也可以被返回:

· DosRequestMutexSem

4 . SHC_START

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 两者均无效)
· ERROR_DATA_ITEM_NOT_SPECIFIED(在流被启动之前必须被联结)

从以下 API 返回的代码也可以被返回:

· DosRequestMutexSem

5 . SHC_STOP

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 两者均无效)
· ERROR_STREAM_NOT_STARTED

从下列 API 返回的代码也可以被返回:

· DosRequestMutexSem
· DosResumeThread
· DosSuspendThread

6 . SHC_SEEK

注意: 如果你要在系统中增加对一种新的非线性数据类型的支持, 则你可以填写一个 MMIO IOProc 来支持 MMIOM_SEEKBYTIME 消息。当文件系统流处理器收到一个

按时间的查找时该消息被发送(mmioSendMessage)。(关于 SpiSeekStream SPI_SEEK-MMTIME 参数的信息请查阅《OS/2 多媒体编程指南》一书。)与此同时,SPCB 通过使 spcb.ulBytesPerUnit 字段或 spcb.mmtimePerUnit 字段为零或两个字段均为零来表明这是一个非线性数据类型。

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 均无效)

· ERROR_NOT_SEEKABLE_BY_TIME

· ERROR_DATA_ITEM_NOT_SEEKABLE

· ERROR_DATA_ITEM_NOT_SPECIFIED

· ERROR_STREAM_NOT_STOP

由以下 API 来的返回代码也可以被返回:

· DosRequestMutexSem

· mmioSeek(源于 mmioGetLastError 的扩展错误代码)

· mmioSendMessage 的 MMIOM_SEEKBYTIME(源于 mmioGetLastError 的扩展错误代码)

7. SHC_GET_PROTOCOL

可能的返回代码有:

· ERROR_INVALID_SPCBKEY

由以下 API 来的返回代码也可以被返回:

· DosRequestMutexSem

8. SHC_INSTALL_PROTOCOL

注意:该流处理器允许安装任何数据类型和子类。

可能的返回代码有:

· ERROR_INVALID_SPCBKEY

· ERROR_ALLOC_RESOURCES

由以下 API 来的返回代码也被返回:

· DosRequestMutexSem

9. SHC_ENUMERATE_PROTOCOLS

可能的返回代码有:

· ERROR_INSUFF_BUFFER

由以下 API 来的返回代码也被返回:

· DosRequestMutexSem

10. SHC_NEGOTIATE_RESULT

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 两者均无效)

· ERROR_INVALID_FUNCTION(必须仅在创建后被直接调用)

由以下 API 来的返回代码也被返回:

· DosRequestMutexSem

A.3.7 被支持的基本流协议控制块数据类型

文件系统流处理器只有一个基本 SPCB,它就是 DATATYPE_GENERIC。如果一个流是由未被安装的 SPCBKEY 创建的,则该流处理器复制最后被安装的 DATATYPE_GENERIC 类型的 SPCB 并使用它。传送的数据类型、子类以及 intkey 被用来取代这些字段的类属性值。基本类属性 SPCB 如表 A-1 示。

表 A-1 文件系统流处理器 SPCB

SPCB 字段	DATATYPE_GENERIC 值
ulBufSize	16KB
ulMinBuf	2
ulMaxBuf	5
ulSrcStart	1
ulTgtStart	1
ulBufFlag	SPCBBUF_NONCONTIGUOUS
ulHandFlag	SPCBHAND_NOSYNC/ SPCBHAND_PHYS_SEEK

A.3.8 流处理器限制

流的最大数目(只受限于可用内存)。

A.4 内存流处理器

有许多多媒体应用程序脚本,其数据流式传输可以向或从系统内存进行。例如,应用程序可以动态地创建一个波形数据对象并且分配应用程序数据缓冲区来存储波形数据。这些数据不是从设备读取而是由应用程序中的算法来产生。这些波形数据可随后被流式传输到波形音频流处理器(目标)。

在另外一个系统内存被用于流式传输的例子中,从文件读取的数据在可以被用作流源之前必须由应用程序进行处理或转换,其中可能包括某些使用了唯一性应用程序数据压缩的情况。在这些情况中,应用程序可以将数据从“平面文件”流式传输到应用程序缓冲区中。接下来,在将这些数据在系统内存中解压缩(或其它操作)以后就可以被执行流式传输(使用内存流处理器作为源),也可能是将这些数据发送到波形音频流处理器。

内存流处理器不支持同步。

内存演播表被用来规定播放或录入的内存地址。

A.4.1 外部接口描述

内存流处理器外部接口之描述如下:

文件名　　MEMSH.DLL
处理器名　MEMSH
处理器类　MEMSYS
PDD/DLL　DLL
源　　　　该流处理器可以是流之源。
目标　　　该流处理器可以是流之目标。

A.4.2　设备控制块

无。

A.4.3　相关控制块

该流处理器支持 ACBTYPE_MEM_PLAYL 类型的相关控制块。

```
/**************************
 * MEMSH - Memory Stream Handler Playlist Object ACB
 **************************/
 #define ACBTYPE_MEM_PLAYL      0x0003L  /* Memory playlist object      */

typedef struct _acb_mem_play1
    {
    ULONG   ulACBLen;          /* Length of structure               */
    ULONG   ulObjType;         /* ACBTYPE_MEM_PLAYL                 */
    PVOID   pMemoryAddr;       /* Starting address of playlist      */
    } ACB_MEM_PLAYL;
```

A.4.4　被支持的隐式(EVENT_IMPLICIT_TYPE)事件

以下对应内存流处理器的隐式(EVENT_IMPLICIT_TYPE)事件是受支持的:

1.EVENT_ERROR

错误类型将被设置在事件隐式控制块(evcb)的 ulFlag 字段。这些错误类型及其标志值被描述于文件系统流处理器一节中(A.3.4 节)。

ulStatus 字段将包含错误代码。被该流处理器产生并返回的可能错误代码有:

- ERROR_PLAYLIST_STACK_OVERFLOW(遭遇了太多的 CALL 指令。最大的 CALL 深度为 20 次调用)
- ERROR_PLAYLIST_STACK_UNDERFLOW(遭遇了太多的 RETURN 指令。对应每一条 CALL 指令只能准确地执行一条 RETURN 指令)
- ERROR_INVALID_FUNCTION(遭遇了无效的演播表操作码)
- ERROR_INVALID_BLOCK(演播表不能读/写访问或数据不能进行读(播放)或写(记录)访问)
- ERROR_END_OF_PLAYLIST(当记录到演播表中时,EXIT 操作码出现在 EOS

之前。流处理器停止)

另外,从以下 API 而来的错误也能被返回:

· DosGetInfoBlock
· DosQueryMem
· SMHEntryPoint SMH_NOTIFY(对应播放的 Give Buf 函数)
· SMHEntryPoint SMH_NOTIFY(对应记录的 GetFull, ReturnEmpty 函数)
· SMHEntry Point SMH_REPORTEVENT

2. EVENT_PLAYLISTCUEPOINT

内存流处理器不接收流中的演播表尾接提示点指示符。当该处理器解释一个 CUEPOINT 演播表操作时它将向流中放置一个尾接提示点指示符。因此该流处理器不产生 EVENT_PLAYLISTCUEPOINT,但它可使后者处于流中供目标流处理器使用。请检查目标流处理器的描述以决定它是否支持 EVENT_PLAYLISTCUEPOINT。

3. EVENT_PLAYLISTMESSAGE

它产生于 MESSAGE 指令被遭遇时。PLAYL_EVCB 的 ulMessageParm 字段将被提供于演播表 MESSAGE 指令的消息参数(第二操作数 operand2)中的消息所填充。ulStatus 字段中包含了演播表指令数。

A.4.5 被支持的显式事件

内存流处理器不支持任何显式事件。

A.4.6 被支持的流处理器命令

以下流处理器命令(SHC)是受支持的。关于这些 SHC 命令及其错误返回代码的描述请查阅《OS/2 多媒体编程参考》一书。

1. SHC_ASSOCIATE

注意:流要求在其可被启动之前将文件打开并与流相联结。在没有停止流的情况下联结一个新对象是不被支持的。

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 均无效)
· ERROR_INVALID_OBJTYPE(只支持 ACBTYPE_MEM_PLAYL)
· ERROR_INVALID_BUFFER_SIZE(ulAcbLen 小于所需)
· ERROR_STREAM_NOT_STOP(流必须被停下来进行联结)
· ERROR_INVALID_BLOCK(内存地址无效——不能读/写访问)

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem
· DosQueryMem

2. SHC_CREATE

注意:如果被传送的 SPCBKEY 不与任何已安装的协议相匹配,则最后一个被安装的带有 DATATYPE_GENERIC 的 SPCB 将被使用。如果 MEMSH 是源,则创建时总是

在 spcb.ulBufFlag 字段中设置 SPCBBUF_USERPROVIDED 标志。

可能的返回代码有:

- ERROR_INVALID_SPCBKEY
- ERROR_ALLOC_RESOURCES

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem
- DosCreateThread

3. SHC_DESTROY

可能的返回代码有:

- ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 均无效)

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem

4. SHC_START

可能的返回代码有:

- ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)
- ERROR_DATA_ITEM_NOT_SPECIFIED(在流被启动之前必须被联结)

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem

5. SHC_STOP

可能的返回代码有:

- ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)
- ERROR_STREAM_NOT_STARTED

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem
- DosResumeThread
- DosSuspendThread

6. SHC_SEEK

注意:不支持向后查找。SEEK_END 仅仅当 lSeekPoint 为 0 时才是可支持的。演播表查找是这样完成的,返回到演播表的开始处并且执行每一条指令,不演播数据但却计算其花费的时间。查找操作于当前演播表,这意味着如果演播表被应用程序修改了则查找使用修改后的演播表来计算查找位置。当执行绝对查找时所有 LOOP 迭代在查找开始计算位置之前被重置为 0。

可能的返回代码有:

- ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)
- ERROR_NOT_SEEKABLE_BY_TIME
- ERROR_DATA_ITEM_NOT_SEEKABLE
- ERROR_STREAM_NOT_STOP
- ERROR_SEEK_BACK_NOT_SUPPORTED

- ERROR_SEEK_PAST_END
- ERROR_INVALID_BLOCK
- ERROR_PLAYLIST_STACK_OVERFLOW
- ERROR_PLAYLIST_STACK_UNDERFLOW
- ERROR_INVALID_FUNCTION
- ERROR_DATA_ITEM_NOT_SPECIFIED

从下列 API 来的返回代码也可被返回:

- DosRequestMutexSem
- DosQueryMem

7 . SHC_GET_PROTOCOL

可能的返回代码有:

- ERROR_INVALID_SPCBKEY

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem

8 . SHC_INSTALL_PROTOCOL

注意: 该流处理器允许安装任何类型和子类。

可能的返回代码有:

- ERROR_INVALID_SPCBKEY
- ERROR_ALLOC_RESOURCES

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem

9 . SHC_ENUMERATE_PROTOCOLS

可能的返回代码有:

- ERROR_INSUFF_BUFFER

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem

10 . SHC_NEGOTIATE_RESULT

可能的返回代码有:

- ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)
- ERROR_INVALID_FUNCTION(必须仅在创建后被直接调用)

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem

A.4.7 被支持的基本流协议控制块数据类型

内存流处理器仅有一个基本流协议控制块(SPCB), 那就是 DATATYPE_GENERIC。如果一个流是由未被安装的 SPCBKEY 创建的, 则该流处理器复制最后被安装的 DATATYPE_GENERIC 类型的 SPCB 并使用它。传送的数据类型、子类以及 intkey 被用来取代这些字段的类属性值。基本类属性 SPCB 如表 A-2 示。

表 A-2　内存流处理器 SPCB

SPCB 字段	DATATYPE. GENERIC 值
ulBufSize	16KB
ulMinBuf	3
ulMaxBuf	10
ulSrcStart	1
ulTgtStart	1
ulBufFlag	SPCBBUF. NONCONTIGUOUS
ulHandFlag	SPCBHAND. NOSYNC/ SPCBHAND. PHYS. SEEK

A.4.8　流处理器限制

流的最大数目(只受限于可用内存)。

A.5　致密盘-数字音频流处理器

许多 CD 上具有包含数字音频的 CD-DA(Compact Disc-Digital Audio, 即致密盘-数字音频, 又称“红皮书音频”)磁道。这些音频磁道可以使用内置式数模转换器(DAC)来播放, 或者这些数据可以被读入系统并且流式传输到带 DAC 硬件 KB 的适配器(例如 AC-PA 卡)。对于后一种情况, CD-DA 流处理器 DLL 是必须的并且充当源流处理器。

该流处理器并非完全运行于实时模式, 但它必须支持连续的数据流式传输。它也不支持同步控制, 因为 CD 设备缺乏实时本质。

该流处理器运行在特定的 CD 地址上。绝大多数 CD-DA 在 CD 的最初大约 2 秒内没有数据, 而某些盘上却有。CD-DA 磁道也可以被安放在混合模式盘上的其它数据磁道之间。因为上述操作, CD-DA 流处理器将盘的绝对开始位置解释为 MMTIME 0。调用程序必须查询盘并且查找所需 CD-DA 磁道的开始位置以及在启动流之前调用 SpiSeek-Stream。应用程序向媒体控制接口编码, 由 CD 音频媒体控制接口驱动程序完成上述操作。

A.5.1　外部接口描述

关于致密盘-数字音频流处理器外部接口的描述如下:

- 文件名　CDDASH.DLL
- 处理器名　CDDASH
- 处理器类　FILESYS
- PDD/ DLL　DLL
- 源　该流处理器可以是流之源。
- 目标　该流处理器可以是流之目标。

A.5.2 设备控制块

无。

A.5.3 相关控制块

该处理器支持 ACBTYPE_CDDA 类型的相关控制块。

```
/ * * * * * * * * * * * * * * * * * * * * * * * * *
 *  CDDASH - CD DA Stream Handler Object ACB
 * * * * * * * * * * * * * * * * * * * * * * * * */
 # define ACBTYPE_CDDA       0x0004L    / * Compact disc - digital audio */
typedef struct _acb_CDDA       / * acbcdda - CD Assoc.Control Block   */
    {
    ULONG  ulACBLen;          / * Length of structure     */
    ULONG  ulObjType;         / * ACB_CDDA                */
    CHAR   bCDDrive;          / * CD drive letter         */
    } ACB_CDDA;
```

A.5.4 被支持的隐式事件(EVENT_IMPLICIT_TYPE)

以下对应 CD-DA 流处理器的隐式(EVENT_IMPLICIT_TYPE)事件是受支持的:

1.EVENT_ERROR

错误类型将被设置在事件隐式控制块(evcb)的 ulFlag 字段中。将被报告的三种错误类型为:

(1) 临时错误(TEMPORARY_ERROR 0x0000L)

在流式传输期间发生了一个错误,但流处理器、同步/ 流管理器或流处理器与同步/ 流管理器两者有能力继续流式传输。

(2) 可恢复错误(RECOVERABLE_ERROR 0x0001L)

发生的错误要求流被停止。应用程序可以在合适时重新启动流。

(3) 不可恢复错误(NONRECOVERABLE_ERROR 0x0002L)

发生了严重错误导致流处理器停止了该流。该流不可能被重新启动。应用程序必须发送一个 SpiDestroyStream 消息。

ulStatus 字段将包含错误代码。能被该流处理器产生并返回的可能错误代码有:

· None

另外,从下列 API 来的错误也可被返回:

· DosGetInfoBlock

· SMHEntryPoint SMH_NOTIFY

· DosDevIOCtl(从 ReadLong 命令到 CD 设备驱动器)。

A.5.5 被支持的显式事件

CD-DA 流处理器不支持任何显式事件。

A.5.6 被支持的流处理器命令

以下流处理器命令(SHC)是受支持的。关于这些 SHC 命令及其错误返回代码的描述请查阅《OS/2 多媒体编程参考》一书。

1. SHC_ASSOCIATE

注意:在流可被启动之前它要求文件被打开并与流相联结。在没有停止流的情况下联结一个新的驱动器符是不被支持的。

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)

· ERROR_INVALID_OBJTYPE(只支持 ACBTYPE_CDDA)

· ERROR_INVALID_BUFFER_SIZE(ulAcbLen 小于所需)

· ERROR_STREAM_NOT_STOP(流必须被停下来进行联结)

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

· DosOpen(针对 CD 设备)

2. SHC_CREATE

注意:唯一可支持的数据类型为 16 位 PCM 立体声,采样频率 44.1kHz。

可能的返回代码有:

· ERROR_INVALID_SPCBKEY

· ERROR_ALLOC_RESOURCES

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

· DosCreateThread

3. SHC_DESTROY

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 均无效)

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

4. SHC_START

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 均无效)

· ERROR_DATA_ITEM_NOT_SPECIFIED(在流被启动之前必须被联结)

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

5. SHC_STOP

· ERROR_INVALID_STREAM(传送的 hstream 无效或 hstream 与 hid 均无效)

· ERROR_STREAM_NOT_STARTED

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

· DosResumeThread

· DosSuspendThread

6 . SHC_SEEK

注意: 请牢记 SEEK 将以绝对方式到达指定地址。从盘的开始位置计算的绝对地址可能与 CD-DA 音乐的开始位置不一样。

另外,查找点对于 SPCB 中指定的 mmtimePerUnit 是粒状的(granular)。

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)

· ERROR_DATA_ITEM_NOT_SEEKABLE

· ERROR_DATA_ITEM_NOT_SPECIFIED

· ERROR_STREAM_NOT_STOP

· ERROR_SEEK_PAST_END

· ERROR_SEEK_BEFORE_BEGINNING

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

· DosDevIOCtl(针对 Seek_End 的 Disk In fo 命令)

7 . SHC_GET_PROTOCOL

可能的返回代码有:

· ERROR_INVALID_SPCBKEY

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

8 . SHC_INSTALL_PROTOCOL

注意: 该流处理器允许安装任何数据类型和子类。

可能的返回代码有:

· ERROR_INVALID_SPCBKEY

· ERROR_ALLOC_RESOURCES

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

9 . SHC_ENUMERATE_PROTOCOLS

可能的返回代码有:

· ERROR_INSUFF_BUFFER

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

10 . SHC_NEGOTIATE_RESULT

可能的返回代码有:

- ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)
- ERROR_INVALID_FUNCTION(必须仅在创建后被直接调用)

从以下 API 来的返回代码也被返回:

- DosRequestMutexSem

A.5.7 被支持的基本流协议控制块数据类型

CD-DA 流处理器只有一个基本 SPCB,它就是带有数据子类型 WAVE_FORMAT_4S16 的 DATATYPE_WAVEFORM。该 SPCB 具有如表 A-3 示的默认值。

表 A-3 致密盘——数字音频流处理器 SPCB

SPCB 字段	DATATYPE_WAVEFORM 值
ulBufSize	48KB
ulMinBuf	8
ulMaxBuf	12
ulSrcStart	1
ulTgtStart	7
ulBufFlag	SPCBBUF_FIXEDBUF
ulHandFlag	SPCBHAND_NOSYNC/ SPCBHAND_PHYS_SEEK
ulBytesPerUnit	588
mmtimePerUnit	10

A.5.8 流处理器限制

流的最大数目(仅受限于可用内存)。

A.6 CD-ROM XA 流处理器

CD-ROM XA 流处理器被用来流式传输交错数据到多个流,因而对应多目标设备。这是从 CD-ROM XA 演播多个流所必须的。例如,一个 CD-ROM XA 文件可能具有需要同时进行流式传输的一个音频信道和一个视频信道。因为音频和视频数据被交错存储于同一文件中,所以,CD-ROM XA 流处理器可以从文件中提取这些单个信道并将其作为独立的源传送给同步/ 流管理器。

A.6.1 外部接口描述

关于 CD-ROM XA 流处理器外部接口的描述如下:

- 文件名　　SSSH.DLL

· 处理器名　　SSSH
· 处理器类　　FILESYS
· PDD/ DLL　　DLL
· 源　　　　　该流处理器可以是流之源。
· 目标　　　　该流处理器不能作为流之目标。

A.6.2　设备控制块

该处理器在 SHC_CREATE 函数上要求标准的 DCB 结构。szDevName 字段必须具有 CD-ROM XA 驱动器的驱动器符。

```
typedef struct _dcb              /* dcb - Device Control Block   */
    {
    ULONG  ulDCBLen;                 /* Length of structure   */
    SZ     szDevName[MAX_SPI_NAME]; /* Device driver name    */
    } DCB;
```

A.6.3　相关控制块

无。

A.6.4　被支持的隐式事件(EVENT_IMPLICIT_TYPE)

以下对应 CD-ROM XA 流处理器的隐式事件(EVENT_IMPLICIT_TYPE)是受支持的:

1.EVENT_ERROR

错误类型将被设置在事件隐式控制块(evcb)的 ulFlag 字段中。将被报告的三种错误类型为:

(1) 临时错误(TEMPOARY_ERROR 0x0000L)

在传式传输期间发生了错误,但流处理器、同步/ 流管理器或流处理器与同步/ 流管理器两者都有能力继续流式传输。

(2) 可恢复错误(RECOVERABLE_ERROR 0x0001L)

发生的错误要求流被停止。在适当时候应用程序可以重新启动流。

(3) 不可恢复错误(NONRECOVERABLE_ERROR 0x0002L)

发生了严重错误导致流处理器停止了该流。流不能够重新启动。应用程序必须发送一条 SpiDestroyStream 消息。

错误代码将包含于 ulStatus 字段中。能被该流处理器产生并返回的可能错误代码有:

· None

另外,从以下 API 来的错误也可被返回:

· DosRequestMutexSem

· DosGetInfoBlocks

· SMHEntryPoint SMH_NOTIFY

· DosDevIOCtl(从 ReadLong 命令到 CD 设备驱动程序)

A.6.5 被支持的显式事件

CD-ROM XA 流处理器不支持任何显式事件。

A.6.6 被支持的流处理器命令

以下流处理器命令(SHC)是受支持的。关于这些 SHC 命令及其错误返回代码的描述请查阅《OS/2 多媒体编程参考》一书。

1. SHC_ASSOCIATE

注意:在流可以被启动之前它要求文件被打开并与流相联结。

可能的返回代码有:

· ERROR_INVALID_STREAM(传送的 hstream 或 hid 无效或两者均无效)

· ERROR_INVALID_OBJTYPE(只支持 ACBTYPE_SSSH)

· ERROR_INVALID_BUFFER_SIZE(ulAcbLen 小于所需)

· ERROR_STREAM_NOT_STOP(流必须被停下来进行联结)

· ERROR_FILE_FORMAT_INCORRECT(文件不是 cdxa 模式 2)

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

· DosDevIOCtl(CD 驱动器上的 ReadLong)

· DosFSCtl(对应 CDFS)

· DosSetFilePtr

2. SHC_CREATE

注意:初级流是为交错文件创建的第一个流。次级流必须在其 SpiCreateStream API 的 hstreambuf 参数中包含初级流的 hstream 句柄。只有对应初级流的 CREATE 需要将驱动器符填入设备控制块(DCB)中。任何在次级流创建时传入的 DCB 都将被该流处理器忽略掉。

可能的返回代码有:

· ERROR_INVALID_SPCBKEY

· ERROR_ALLOC_RESOURCES

· ERROR_TOO_MANY_STREAMS(每个 XA 文件只允许 16 个流)

· ERROR_INVALID_BUFFER_SIZE(DCB 太小)

· ERROR_INVALID_PARAMETER(DCB 参数为空)

从以下 API 来的返回代码也被返回:

· DosRequestMutexSem

· DosCreateThread

· 对应 CD 驱动器的 DosOpen

3 . SHC_DESTROY

注意:破坏初级流将会挂起次级流的流式传输,因为所有被联结的流都使用由初级流所拥有的缓冲区。

可能的返回代码有:

· ERROR_INVALID_STREAM(**传送的** hstream **或** hid **无效或两者均无效**)

从以下 API **来的返回代码也被返回:**

· DosRequestMutexSem

4 . SHC_START

注意:对于初级流,只有当 SHC_START **命令被接收到时才启动流式传输,因为它拥有** I/O **线程和流缓冲区。**

可能的返回代码有:

· ERROR_INVALID_STREAM(**传送的** hstream **或** hid **无效或两者均无效**)

· ERROR_DATA_ITEM_NOT_SPECIFIED(**在流被启动之前必须被联结**)

从以下 API **来的返回代码也被返回:**

· DosRequestMutexSem

5 . SHC_STOP

注意:停止初级流将会停止所有的流,因为初级流拥有 I/O **线程和缓冲区。停止次级流将仅仅停止对应该流的数据。如果在初级流正在运行时次级流被停止并且重新启动,则次级流将从初级流所在的交错点开始拾取数据。**

可能的返回代码有:

· ERROR_INVALID_STREAM(**传送的** hstream **或** hid **无效或两者均无效**)

· ERROR_STREAM_NOT_STARTED

从以下 API **来的返回代码也被返回:**

· DosRequestMutexSem

· DosResumeThread

· DosSuspendThread

6 . SHC_SEEK

注意: SEEK **只对初级流有效。对次级流执行** SEEK **命令将返回** ERROR_DATA_ITEM_NOT_SEEKABLE。

查找点对于 SPCB **中指定的** mmtimePerUnit **为粒状的**(granular)。

可能的返回代码有:

· ERROR_INVALID_STREAM(**传送的** hstream **或** hid **无效或两者均无效**)

· ERROR_DATA_ITEM_NOT_SEEKABLE

· ERROR_DATA_ITEM_NOT_SPECIFIED

· ERROR_STREAM_NOT_STOP

· ERROR_SEEK_PAST_END

· ERROR_SEEK_BEFORE_BEGINNING

· ERROR_LARGE_SEEK_BY_TIME

从以下 API **来的返回代码也被返回**:

· DosRequestMutexSem

· DosDevIOCtl(ReadLong **命令至** CD **设备驱动程序**)

7 . SHC. GET. PROTOCOL

可能的返回代码有:

· ERROR. INVALID. SPCBKEY

从以下 API **来的返回代码也被返回**:

· DosRequestMutexSem

8 . SHC. INSTALL. PROTOCOL

注意: **该流处理器只允许安装数据类型** DATATYPE. CDXA. AUDIO, DATATYPE. CDXA. VIDEO **或** DATATYPE. CDXA. DATA。

可能的返回代码有:

· ERROR. INVALID. SPCBKEY

· ERROR. ALLOC. RESOURCES

从以下 API **来的返回代码也被返回**:

· DosRequestMutexSem

9 . SHC. ENUMERATE. PROTOCOLS

可能的返回代码有:

· ERROR. INSUFF. BUFFER

从以下 API **来的返回代码也被返回**:

· DosRequestMutexSem

10 . SHC. NEGOTIATE. RESULT

可能的返回代码有:

· ERROR. INVALID. STREAM(**传送的** hstream、hid **或两者均无效**)

· ERROR. INVALID. FUNCTION(**必须仅在创建后被直接调用**)

从以下 API **来的返回代码也被返回**:

· DosRequestMutexSem

A .6 .7 被支持的基本流协议控制块数据类型

CD-ROM XA **流处理器有七个基本** SPCB, **如表** A-4—**表** A-7 **示**。

表 A-4 CD-ROM XA 流处理器 SPCB: CDXA. LEVELB,CDXA. LEVEC

SPCB 字段	CDXA. LEVELB 值	CDXA. LEVELC 值
spcbkey .ulDataType	DATATYPE. CDXA. AUDIO	DATATYPE. CDXA. AUDIO
spcbkey .ulDataSubType	CDXA. LEVELB	CDXA. LEVELC
ulDataFlag	SPCBDATA. CUETIME	SPCBDATA. CUETIME
ulNumRec	17	17

续表

SPCB 字段	CDXA. LEVELB 值	CDXA. LEVELC 值
ulBlockSize	1	1
ulBufSize	39984	39984
ulMinBuf	8	8
ulMaxBuf	16	16
ulSrcStart	1	1
ulTgtStart	1	1
ulBufFlag	SPCBBUF. INTERLEAVED	SPCBBUF. INTERLEAVED
ulHandFlag	SPCBHAND. NOSYNC/ SPCBHAND. PHYS. SEEK	
ulBytesPerUnit	2304	2304
mmtimePerUnit	160	320

表 A-5　CD-ROM XA 流处理器 SPCB：LEVELB. MONO,LEVELC. MONO

SPCB 字段	LEVELB. MONO 值	LEVELC. MONO 值
spcbkey .ulDataType	DATATYPE. CDXA. AUDIO	DATATYPE. CDXA. AUDIO
spcbkey .ulDataSubType	CDXA. LEVELB. MONO	CDXA. LEVELC. MONO
ulDataFlag	SPCBDATA. CUETIME	SPCBDATA. CUETIME
ulNumRec	17	17
ulBlockSize	1	1
ulBufSize	39984	39984
ulMinBuf	8	8
ulMaxBuf	16	16
ulSrcStart	1	1
ulTgtStart	1	1
ulBufFlag	SPCBBUF. INTERLEAVED	SPCBBUF. INTERLEAVED
ulHandFlag	SPCBHAND. NOSYNC/ SPCBHAND. PHYS. SEEK	
ulBytesPerUnit	2304	2304
mmtimePerUnit	320	640

表 A-6　CD-ROM XA 流处理器 SPCB：CDXA. AUDIO. HD

SPCB 字段	CDXA. AUDIO 值
spcbkey .ulDataType	DATATYPE. CDXA. AUDIO
spcbkey .ulDataSubType	CDXA. AUDIO. HD
ulDataFlag	SPCBDATA. CUETIME
ulNumRec	17
ulBlockSize	1
ulBufSize	39984
ulMinBuf	8
ulMaxBuf	16
ulSrcStart	1
ulTgtStart	1
ulBufFlag	SPCBBUF. INTERLEAVED
ulHandFlag	SPCBHAND. NOSYNC/ SPCBHAND. PHYS. SEEK
ulBytesPerUnit	0
mmtimePerUnit	0

表 A-7　CD-ROM XA 流处理器 SPCB：CDXA. DATA，CDXA. VIDEO

SPCB 字段	CDXA. DATA 值	CDXA. VIDEO 值
spcbkey .ulDataType	DATATYPE. CDXA. DATA	DATATYPE. CDXA. VIDEO
spcbkey .ulDataSubType	0	0
ulNumRec	17	17
ulBlockSize	1	1
ulBufSize	39984	39984
ulMinBuf	8	8
ulMaxBuf	16	16
ulSrcStart	1	1
ulTgtStart	1	1
ulBufFlag	SPCBBUF. INTERLEAVED	SPCBBUF. INTERLEAVED
ulHandFlag	SPCBHAND. NOSYNC/ SPCBHAND. PHYS. SEEK	
ulBytesPerUnit	0	0
mmtimePerUnit	0	0

A.6.8 流处理器限制

同一个初级流相联结的流的最大数目为 15。

在同一时刻被该流处理器所支持的流的最大数目仅受限制于可用内存。

附录 B　P2STRING 工具

P2STRING 命令组处理工具被用来在 OS/ 2 多媒体环境下测试媒体控制接口字符串命令。P2STRING 工具通过对命令组文件(包含字符串命令和工具伪指令)进行处理来测试用户应用程序中的字符串命令。P2STRING 从命令组(script)文件中抽取字符串并且通过 mciSendString 函数对命令进行处理。包含在命令组中的进程消息和错误状态被记录到一个输出文件中并且显示在窗口内(如图 B-1 所示)。每一个进程记录到其自身的显示窗口,但所有的进程都记录到同一输出文件中。此外,如果用户关闭了显示窗口,则字符串执行线程立即被破坏掉并且整个进程结束。

图 B-1　P2STRING 显示窗口

输出信息包括所有从命令组文件中读出的非注释行(例如描述伪指令、命令字符串、预期返回值、预期和接收的通知消息以及状态行)。此外,不成功字符串命令的错误和调试语句被记录到一个名为 P2STRING .LOG 的文件中。

B .1　设置字体尺寸和类型

在启动P2STRING工具之前,用户可以改变显示于台面P2STRING窗口中的字体尺寸和类型。例如,要指定字体尺寸为10点的Times罗马字体则如图B-2示,可以键入:

```
SET P2STRING.FONTFACE = TIMES
SET P2STRING.FONTSIZE = 10
```

图 B-2　字体尺寸和类型设置示例

用户可以指定字体字面或字体尺寸或两者都指定。可能的 FONTFACE 值包括 SYSTEM,COURIER,TIMES 或 HELVETICA** (默认)。可能的 FONTSIZE 值包括 8 (默认),10,12,14 或 18。

B.2　启动 P2STRING

P2STRING 工具包括两个文件: P2STRING.EXE 和 P2S_DLL.DLL。这些文件位于子目录 \ TOOLKIT \ BIN \ BETA \ P2STRING 中。图 B-3 显示的句法被用来启动 P2STRING 程序。相关参数描述于表 B-1 中。

```
P2STRING inp_file [ - a] out_file [ - eerr_file][ - d| - D][ - E][ - t]
```

图 B-3　P2STRING 句法图

注意: 这些参数是区分大小写的。

表 B-1 描述了与 P2STRING 程序相关的参数。

表 B-1　P2STRING 参数

参　数	描　　述
script_file	指定用户要处理的命令组文件名。 注意: 关于命令组文件内容和如何解释命令组语言的信息请查阅 P2STRING Script Language(《P2STRING 命令组语言》)。
[- a]out_file	指定包含测试结果的输出文件名。除非用户指定了可选参数[- a],否则该文件将只包含用户正在运行的测试结果。例如,为了将一个名为 SAMPLE.SCR 的命令组文件的输出结果拼接到名为 MDM.OUT 的当前输出文件中,应键入: P2STRING SAMPLE.SCR - aMDM.OUT
[- eerr-file]	指定可选的错误文件名来接收那些未被成功地完成的字符串命令所发出的消息。例如要创建一个名为 MDM.ERR 的错误文件可键入: P2STRING SAMPLE.SCR MDM.OUT - eMDM.ERR
[- d/ - D]	指定以下可选参数中的一个: · - d　指示 P2STRING 在处理完命令组文件以后终止运行。当用户正在自动运行测试实例时可以使用该参数,它不会改变输出结果。当命令组文件已完成处理时 P2STRING 用一条要求终止测试的消息来提示用户。 · - D　其行为类似于 - d 参数,除了要求用户输入的描述伪指令被忽略以外。 注意: 关于如何在一个命令组文件中加入执行伪指令(它要求用户输入)的信息请查阅本附录 B.3.2 节内容"工具伪指令"。

续表

参　数	描　　述
- E	促使命令组文件在第一个错误发生后退出。默认情况下,命令组文件一直运行到结束而不考虑错误。例如以下命令将在遇到一个错误后结束 SAMPLE .SCR 的处理: P2STRING SAMPLE .SCR - aMDM .OUT - d - eMDM .ERR - E
- t	记录字符串的时间标记以及 MM. MCIPASSDEVICE 通知消息。

B .3　P2STRING 命令组语言(Script Language)

这一节将描述命令组文件的内容以及如何解释命令组语言。命令组文件可以包含以下类型的行:

· 注释

· 工具伪指令

· OS/ 2 多媒体字符串命令

· 预期的返回字符串

· 预期的错误消息

· 预期的通知消息

以下讨论所提供的信息告诉用户如何创建一个包含这些不同行类型的命令组文件。图 B-4 显示了一个作为示例的命令组文件。

```
@PROCESSES = 2
@EVENTS = {HASCTRL1 = 1,HASCTRL2 = 0}
#
#
#
@PROCESS 1
;
; set masteraudio level for session to 10% - will affect all
; 3 processes
;
masteraudio volume 10
;
; open waveaudio device non-exclusively
;
open waveaudio alias wav1 shareable notify
+ MM_MCINOTIFY MCI_NOTIFY_SUCCESSFUL MCI_OPEN #1
@WAIT_NOTIFY 1 60000
@WAIT_PASSDEVICE wav1 60000
```

```
@WAIT_NOTIFY 21 60000
```

图 B-4　命令组文件样例

B.3.1　注释

命令组注释行必须以分号(;)或镑号　(#)作为开始的第一列。在输出文件中这些注释行既不显示也不回送。如果用户需要将一个标记显示在输出文件中,请使用@REM伪指令。

P2STRING 允许在其窗口中显示可变数量的行。正规的注释行(标题行)既不被显示也不写入输出文件。

B.3.2　工具伪指令

P2STRING 工具支持多线程或多进程的执行,但不同时支持两者。用户可以在命令组文件中使用工具伪指令来测试@PROCESSES 或@THREADS。

工具伪指令以位于第一列的位于符号@开始。这些伪指令将影响输出的执行和外观。以下类的伪指令是可识别的:

· Initialization(初始化)

· Execution(执行)

B.3.2.1　初始化伪指令

使用初始化伪指令可以设置命令组文件的内容。这些伪指令必须出现在执行伪指令之前,因为该工具预先处理命令组文件并且建立进程命令缓冲区。

@THREADS 和@PROCESSES 伪指令是互斥的。换句话说,P2STRING 工具支持命令组文件中的或者多线程或者多进程执行(非两者同时)。此外,每一个命令组文件限制最多有 10 个进程或线程。

表 B-2 列举了可支持的初始化伪指令。

表 B-2　初始化伪指令

伪指令	描　　述
@PROCESSES = x	指定将被运行的命令组文件中的进程数目(x)。例如: @PROCESSES = 2
@THREADS = x	指定将被运行的命令组文件中的线程数目(x)。
@EVENTS = {n[= 0/ 1] [, n[= 0/ 1]] }	指定一个或多个事件名。事件由用户定义,最多为 15 个字符。事件可被设置为 0 或 1。对于那些由@SET_EVENT 执行伪指令设置的事件,设置事件为 1。设置事件为 0 将清除或重新设置该事件。如果没有指定初始化值则事件被初始化为 0。例如: @EVENTS = {brad = 1 , john = 1 , test = 0}

B.3.2.2　执行伪指令

执行伪指令被用来对命令组文件进行处理。@THREAD 和@PROCESS 伪指令还是

互斥的。

表 B-3 列举了被支持的执行伪指令。

注意：在@WAIT. EVENT 和@WAIT. NOTIFY 伪指令上的超时不会被认为是失败。

表 B-3 执行伪指令

伪指令	描　　述
@THREAD x	指定从该伪指令开始直到遭遇下一个@THREAD 伪指令之间的命令组行属于线程号 x。
@PROCESS x	指定从该伪指令开始直到遭遇下一个@PROCESS 伪指令之间的命令组行属于进程号 x。
@SET. EVENT name 0/ 1	将事件 name 设置为 0 或 1。用 1 来标记事件已发生,用 0 来清除或重新设置事件。 注意：事件必须通过伪指令@EVENTS 加以声明。
@WAIT. EVENT name[to]	等待直到事件 name 被设置为 1。@WAIT. EVENT 并不促成事件状态的变更。如果用户需要一个可重复使用的事件,则可使用这个伪指令。 时限(to)被指定以毫秒为单位。如果省略该项,则默认为 3 分钟。
@WAIT. NOTIFY x[to]	等待一个从预期的 MM. MCINOTIFY 通知消息来的特定数字(x)。该数字必须与 MM. MCINOTIFY 参考行中所使用的索引相符合。@WAIT. NOTIFY 事件是不可重复使用的,因为从逻辑上没有任何事件来重新设置它。 时限(to)被指定以毫秒为单位。如果省略该项,则默认为 3 分钟。 如果被联结的 mciSendString 函数失败,则该事件被公布以阻止对不再可能被发送的通知的时间延迟。 注意：关于 MM. MCINOTIFY 通知消息的信息请参看本附录 B3.6.1 节“命令完成/ 错误通知”。
@WAIT. PASSDEVICE alias [to]	等待直到别名为 alias 的设备实例获得使用。这里假设使用在命令组文件中的别名是唯一的。最大别名长度为 20 个字符。 注意：对于每一个 OPEN 命令使用唯一的别名。 时限(to)被指定以毫秒为单位。如果忽略该项,则默认为 3 分钟。 P2STRING 工具假设每个 OPEN 字符串命令被指定了一个唯一的别名。如果不使用唯一的别名,则错误状态就可能发生。
@REM comment	将注释语句 comment 回显到屏幕和输出记录文件中。而所有其它的命令组注释行(以“;”或“#”开始者)则既不传送也不被显示。

续表

伪指令	描　述
@PAUSE to	在指定时间内暂停处理输入命令组文件中的当前线程或进程。它不停止通知或窗口函数的处理。其它线程或进程不受该伪指令影响。to 为暂停时间,单位为毫秒。
@BREAK [message]	促使一个消息框来显示 message 文本。命令组处理被中断直到用户以正确的动作进行响应。例如: @BREAK [replace the CD]
@CHEAK [message]	在用户响应的基础上对先前命令的成功进行分类。用一个弹出窗口来显示 message 并且用 Yes 或 No 按钮来提示用户。如果用户选择 Yes 则状态被传送,否则如果选择 No 则不传送。例如: @BREAK The music will play for 5 secs . Ready to time it ? play cdaudio notify @PAUSE 5000 @CHECK Did it play ?

B.3.3 OS/ 2 多媒体字符串命令

不属于任何其它类型的所有行均被解释为 OS/ 2 多媒体字符串命令。在其环境变量被取代以后,这些行被通过 mciSendString 函数传送到媒体设备管理器(MDM)。

在字符串命令行中被问号所括起来的任何标记(诸如 ? FOO ? 之类)都被解释为环境变量。在其被传送到 mciSendString 函数之前环境变量的实际值被代入字符串中。如果该变量没有被发现,则发送一条警告并且该标记被 NULL 字符串所取代。例如,假定环境字符串 MMDATA 被设置为 D \ DATA,则 open ? mmdata ? \ temp .wav alias a 等价于 open d \ data \ temp .wav alias a。

B.3.4 预期的返回字符串

许多 OS/ 2 多媒体命令返回字符串。利用预期返回字符串行根据一个期望值来检验这些字符串是可能的。

预期返回字符串行具有以下格式:

= result

等号(=)必须位于第 1 列并且不应当有尾随空格。如果期待一个空字符串,则在 = 后不跟随任何字符(空格也不允许)。例如:

```
status cdaudio ready wait
 = TRUE
status cdaudio mode wait
 = stopped
```

预期的结果总是被解释为字符串。对于那些返回二进制数据的命令来说,这样作将

有可能产生某些不寻常的输出。

在返回文本数字值的地方,通过在数字之前使用代字号(～)可以给它匹配一个±10%的容差范围。当准确值无从得知时这是一种典型用法。例如:

```
set foo time format milliseconds wait
play foo notify
@PAUSE 1000
stop foo wait
status foo position wait
=1000
```

因此,如果返回字符串在900—1100的范围之间状态命令就被认为是成功的。

B.3.5 预期的错误消息

当一个OS/2多媒体字符串命令被期望以一个错误失败时,可以使用预期错误行来指定预期的错误。预期错误行具有以下格式:

=!error

=!必须从第1列开始。如果任何错误都是可接受的,则可使用关键字ERROR。如果期待一个特定的错误,则可在=!后键入准确的错误消息。例如:

```
open sequencer alias mymidi wait
load mymidi nofile.foo
=!File not found.
```

在预期错误和预期结果行中都要小心附加的间隔。字符串的大小写并不重要,然而如果间隔和标点不能正确地匹配则比较将会失败。

B.3.6 预期的通知消息

许多OS/2多媒体字符串命令促使通知消息被发送到P2STRING工具。系统使用通知消息来对应用程序进行响应。例如,通知消息可以指示关于媒体设备函数的完成或媒体控制设备所有权从一个进程到另一个进程的传递等系统状态。

通过使用预期通知行来验证正确的通知被接收到是可能的。每一预期通知行以位于第一列的加号(+)开始。以下类型的通知行是可能的:

· 命令完成/错误通知

· 事件通知

· 位置变更通知

注意:对于单个OS/2多媒体字符串命令来说,任何或所有通知消息都是可以预期的。

B.3.6.1 命令完成/错误通知

当设备成功地完成了由媒体消息所指示的动作或发生了错误时MM_MCINOTIFY

行将通知应用程序。这种通知行的格式如下:

```
+ MM_MCINOTIFY notify - code message [ # x]
```

其关键字的含义见于表 B-4。

表 B-4　命令完成/ 错误通知

关键字	描　述
notify - code	指定通知消息代码, 例如 MCI_NOTIFY_SUCCESSFUL。其拼写必须与 OS2ME .H 文件中的 # defines 语句所定义的通知代码相同
message	指定引起通知的媒体控制接口消息, 例如 MCI_PLAY。其拼写必须与 OS2ME .H 文件中的 # defines 语句所定义的 MCI 消息相同
x	指定一个唯一的数字 (x) 用来将@ WAIT_NOTIFY 伪指令与特定通知相联结。该数字与字符串发送次序无关,它必须是唯一的并且在 1—99 范围内

B .3 .6 .2　事件通知

MM_MCIPOSITIONCHANGE 行将把当前媒体位置通知给应用程序。其格式如下:

```
+ MM_MCIPOSITIONCHANGE position % user-parameter # x
```

句法中各项关键字的含义见于表 B-5。

表 B-5　事件通知

关键字	描　述
position	指定第一个位置变更消息的预期 MMTIME 位置。
user - parameter	指定被返回的用户参数。这将与 SETPOSITIONADVISE 字符串命令中指定的返回值相同。注意返回值必须是一个在 1—99 范围内的唯一整数。
x	指定预期的位置变更消息的数目(x)。关于进一步的信息请参看本附录 B .3 .7 节内容“ MM_MCIPOSITIONCHANGE 验证的限制”。

B .3 .6 .3　位置变更通知

MM_MCICUEPOINT 行将通知应用程序演播表处理器已遭遇了一个 message 指令。其格式如下:

```
+ MM_MCICUEPOINT position % user-parameter
```

句法中各项关键字的含义见于表 B-6。

表 B-6　位置变更通知

关键字	描述
position	指定第一个位置变更消息的预期 MMTIME 位置。
user-parameter	指定被返回的用户参数。这将与 SETPOSITIONADVISE 字符串命令中所指定的返回值相同。注意返回值必须是在 1—99 范围内的唯一整数。

注意: 其它通知不可能被比较,因为它们不允许将用户参数作为消息的一部分,而该参数对于相关通知的跟踪是必不可少的。

B.4　MM_MCIPOSITIONCHANGE 验证的限制

用户在 P2STRING 工具中使用事件通知行所能验证的事项是有限制的。MM_MCIPOSITIONCHANGE 行要求在各自的字符串命令中使用"返回值"项。该行也不提供计时起始点,如演播开始点。P2STRING 工具只能计算特定用户参数(被用作关键字)所接收的消息数目并且检查随后的消息是否具有近似于特定预期位置间隔的位置。命令组编写者必须决定有多少 SETPOSITIONADVISE 消息被预期,考虑演播的持续时间、时间格式以及演播/记录的开始位置。给定于预期通知行中的参考位置必须以 MMTIME 为单位。如果"预期消息数目"参数被省略,则工具只验证位置间隔(不是数目)。假如命令组中的演播(play)、查找(seek)或记录(record)被用来覆盖非单调范围,则 P2STRING 可能会报告位置建议(position-advises)上的失败,因为它期望每一个 SETPOSITIONADVISE 处在相对于前一条信息的下一个位置间隔上。如果在命令组中作了一次跳跃到达一个与此不符的位置,则状态将被记录为失败。

例如:

```
setpositionadvise SomeDevice every 10000 on return 5
 + MM_MCIPOSITIONCHANGE 10000 %5
play SomeDevice from 35000 to 55000 notify (produce 3 positionchange msgs)
seek SomeDevice to 30000 wait
play SomeDevice notify (produces a number of messages starting at 30000)
```

MM_MCIPOSITIONCHANGE 消息被记录为失败,因为在位置间隔上有所偏离。处理这种情况的一种办法是在显式位置跳跃被执行之前使 MM_MCIPOSITIONCHANGE 无效,而在执行跳跃以后再以不同的用户参数使相同的 SETPOSITIONADVISE 起作用。

例如:

```
setpositionadvise SomeDevice every 10000 on return 5
 + MM_MCIPOSITIONCHANGE 10000 %5
play SomeDevice from 35000 to 55000 notify (produce 3 positionchange msgs)
```

```
setpositionadvise SomeDevice every 10000 off
seek SomeDevice to 30000 wait
setpositionadvise SomeDevice every 10000 on return 6
 + MM_MCIPOSITIONCHANGE 10000 % 6
play SomeDevice notify (produce a number of messages starting at 30000)
```

B.5 处理逻辑

一个字符串命令可以跟随有 0 到 1 个返回值行或者 0 到 3 个通知行。注意此处有一个排它性的“或者”,这就意味着同时指定预期返回值和预期通知将不会给出可信的结果,其原因在于这样一个事实的存在,即在通知消息结束以前返回缓冲区不会变为有效。另外,如果通知标志、返回值处于过程接口格式而不是字符串接口格式则也会出错。

MDM 将不可能替以通知标志进行处理的命令来将返回值转换成字符串,因为媒体控制驱动程序将正在把它们的通知消息直接发送到应用程序。

每一个命令的状态取决于两个阶段。第一个阶段是在字符串执行时。如果 mciSendString 函数返回了一个错误并且在命令组中不存在跟随该命令字符串行的 = ! ERROR 参考行,则该命令被考虑为失败。如果在 mciSendString 被处理以后发现了一个返回值,则工具将检查预期返回值并对两者进行比较。如果它们不相符合,则该命令被考虑为失败。如果所发生的错误超出了 mciGetError 函数所能理解的范围,则即使遇到了 ! ERROR,该命令也被考虑为失败。

比较的第二个阶段是在通知被接收到以后以及所有命令都被发送以后。如果通知被接收到并且成功地与预期通知行进行了比较,则该命令被考虑为成功。如果不存在参考通知行,则命令状态将不被确定,除非返回的是错误通知。当所有的命令组被处理完以后,预期参考通知将被用来决定是否所有的通知都被接收。那些没有接收到通知并且具有预期通知行类型的命令将被标记为失败。请注意命令字符串并不检查通知标志的存在性,创建一个预期通知行是编写者的责任,如果该预期通知行很重要的话。对于预期的 NOTIFY_SUCCESSFUL 消息,所有代码,除了 NOTIFY_ERROR 以外,都被认为是有效的。这包括 NOTIFY_SUCCESSFUL, NOTIFY_ABORTED 以及 NOTIFY_SUPERCEDED。如果任何其它的通知代码被指定为预期通知,则将检查其准确匹配性。如果 NOTIFY_ERROR 被预期,则范围内的所有错误都将被验证为通过。这里不存在任何手段来验证通知中所返回的准确错误代码。

附录C 通　　告

本出版物中对 IBM 产品、程序或者服务的引用并不表明 IBM 公司试图使它们可用于 IBM 运作的所有国家。任何对 IBM 产品、程序或者服务的引用并不打算声明或暗示只能使用 IBM 产品、程序或服务。任何功能上等价的产品、程序或服务,只要不侵犯 IBM 公司的知识产权或其它合法保护权,都可以被用来取代 IBM 产品、程序或服务。除了那些由 IBM 明确指出的以外,与其它产品、程序或服务共同评估和验证操作性能是用户的责任。

IBM 公司对于本文献中所包含的题材可能拥有专利或悬而未决的专利申请。本文献的篇幅中并没有给出这些专利的任何证书。需要发送证书质询的用户可写信给本公司证书总监,地址: IBM Director of Licensing, IBM Corporation, 500 Colubus Avenue, Thornwood NY 10594, U.S.A。

C.1 商　　标

以下在本出版物中标注有星号(*)的词条是 IBM 公司在美国或其它国家的商标:

Audio Visual Connection

IBM

Common User Access

CUA

Multimedia Presentation Manager/ 2

OS/ 2

Presentation Manager

Workplace Shell

以下在本出版物中标注有双星号(* *)的词条是如下的其它公司的商标。另外的商标则是其各自公司的商标。

Helvetica	Linotype Company
Pro AudioSpectrum 16	Media Vision, Inc.
Sound Blaster	Creative Labs, Inc.